



**António Capôto Relvas**

Licenciatura em Engenharia Informática

## **Uma Interface Web para apoio à deteção de Concerns em código MATLAB**

Dissertação para obtenção do Grau de Mestre em  
**Engenharia Informática**

Orientador: Nuno Miguel Cavalheiro Marques, Professor Auxiliar,  
Faculdade de Ciências e Tecnologia  
da Universidade Nova de Lisboa  
Co-orientador: Miguel Pessoa Monteiro, Professor Auxiliar, Facul-  
dade de Ciências e Tecnologia  
da Universidade Nova de Lisboa

Júri

Presidente: Prof<sup>o</sup> Doutora Fernanda Barbosa  
Vogais: Prof. Doutor Luiz Manuel Pereira Sales Cavique Santos  
Prof. Doutor Nuno Miguel Cavalheiro Marques



FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE NOVA DE LISBOA

**Junho, 2019**



## **Uma Interface Web para apoio à deteção de Concerns em código MATLAB**

Copyright © António Capôto Relvas, Faculdade de Ciências e Tecnologia, Universidade NOVA de Lisboa.

A Faculdade de Ciências e Tecnologia e a Universidade NOVA de Lisboa têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.



## AGRADECIMENTOS

Gostaria de agradecer ao meu orientador, professor Nuno Marques, pelas reuniões em que discutimos ideias e estratégias de atacar os problemas bem como de todas as outras funções enquanto orientador. Ao meu co-orientador, professor Miguel Monteiro, por toda a grande disponibilidade que demonstrou em me apresentar os conceitos do problema e tirar quaisquer dúvidas que tivesse.

Agradeço à Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, em especial ao Departamento de Informática e aos seus docentes por todo o conhecimento transmitido.



## RESUMO

---

Linguagens de manipulação de dados e de matrizes são ferramentas essenciais para analisadores de dados. Contudo verifica-se que estas não conferem o suporte à modularidade que seria desejável.

Este trabalho apresenta um protótipo web que recorre a modelos produzidos pelo o algoritmo Mapa Auto-Organizado (em Inglês: *Self-Organizing Map* - SOM) sendo o acrónimo em inglês SOM, para estudar os problemas de modularidade presentes em código *MATLAB*. Para este efeito, a visualização de dados é fundamental na análise dos mesmos. O SOM usualmente disponibiliza dois tipos de representação: as *Component Planes* e a *U-Matrix*. Todavia, estas são representações complexas que sem um conhecimento profundo sobre as mesmas, o qual não é trivial de obter, tornam difícil a compreensão dos resultados apresentados. Dessa forma, é necessário estudar soluções que permitam a compreensão fácil dos resultados produzidos pelos SOM. Esta dissertação tem como um dos focos, estudar metáforas visuais de apoio à compreensão dos dados por parte de programadores interessados em linguagens orientadas ao processamento de matrizes. Sendo o outro foco principal a validação do SOM como ferramenta de análise exploratória através das metáforas implementadas. Em particular, 3 vistas foram desenvolvidas para programadores com conhecimento da linguagem *MATLAB*. Uma base de dados foi concebida para armazenar dados sobre código *MATLAB*, a serem usados como base para análise e respectiva produção de resultados. É apresentado um sistema de anotações, através do qual os utilizadores especialistas em SOM e em análise de código, anotam código *MATLAB* e partes de metáforas visuais, de forma a que utilizadores não especialistas conseguem compreender os resultados apresentados com mais facilidade - e dessa maneira tirar o máximo proveito das análises. Um conjunto de funcionalidades foi desenvolvidas à volta deste conceito de forma a potenciar a sua utilidade.

**Palavras-chave:** MatLab, Concerns, Tokens, SOM, UbiSOM, Metáforas Visuais, Aplicação Web

---





## ABSTRACT

---

Data manipulation and matrix manipulation languages are essential tools for data analyzers. However, it is verified that they do not offer the desirable modularity that would be desirable. This work presents a web prototype that makes use of models produced by the Self-Organizing Map (SOM) algorithm, being the acronym in English SOM, to study the modularity problems present in MATLAB code. For this purpose, data visualization is fundamental in their analysis. SOM usually provides two types of representation: the Component Planes and the U-Matrix. However, these are complex representations that without a thorough knowledge about them, which is not trivial to obtain, make it difficult to understand the results presented. Thus, it is necessary to study solutions that allow easy understanding of the results produced by SOM. This thesis has as one of the focuses, to study visual metaphors to support the understanding of the data by programmers interested in languages oriented to matrix processing. The other main focus is the validation of SOM as an exploratory analysis tool through the implemented metaphors. In particular, 3 views were developed for programmers with knowledge of the MATLAB language. A database was designed to store data on MATLAB code, to be used as the basis for analysis and its production of results. An annotation system is presented, whereby specialist users of SOM and MATLAB code analysers, annotate MATLAB code and parts of visual metaphors, so that non-expert users can understand the results presented more easily - and thus take the maximum benefit of the analyses. A set of functionalities has been developed around this concept in order to enhance its usefulness.

**Keywords:** MatLab, Concerns, Tokens, SOM, UbiSOM, Visual Metaphors, Web Application

---



# ÍNDICE

<b>Lista de Figuras</b>	<b>xv</b>
<b>Lista de Tabelas</b>	<b>xvii</b>
<b>Listagens</b>	<b>xix</b>
<b>1 Introdução</b>	<b>1</b>
1.1 Objetivos . . . . .	2
1.2 Contribuições . . . . .	3
1.3 Estrutura do Documento . . . . .	3
<b>2 Modularidade e Concerns</b>	<b>5</b>
2.1 Modularidade . . . . .	5
2.2 Concerns . . . . .	6
2.3 CrossCutting Concerns . . . . .	6
2.3.1 CCC em MATLAB . . . . .	7
2.4 Abordagem para a detecção de concerns em MATLAB . . . . .	8
2.4.1 Mineração de Aspetos . . . . .	8
2.4.2 Token e Word Token . . . . .	9
2.4.3 Mineração de concerns baseada em tokens . . . . .	9
2.4.4 Métricas . . . . .	10
2.5 CCCExplorer: ferramenta de extracção de métricas . . . . .	12
<b>3 SOM, Visualização e Metáforas Visuais</b>	<b>17</b>
3.1 Mapas Auto Organizados . . . . .	17
3.1.1 Fase de Treino dos Mapas . . . . .	18
3.1.2 Interpretação do SOM . . . . .	19
3.2 Atributos Reais e Atributos Visuais . . . . .	20
3.3 SourceMiner . . . . .	21
3.4 Visualização . . . . .	21
3.5 Metáforas e Visualização Exploratória . . . . .	22
3.5.1 Categorização das Metáforas Visuais . . . . .	23
3.6 Técnicas de Interação . . . . .	24

3.6.1	Filtragem Interativa . . . . .	24
3.6.2	Zoom Interativo . . . . .	24
3.6.3	Distorção Interativa . . . . .	25
3.7	Múltiplas Perspectivas e Múltiplas Visões . . . . .	25
3.7.1	Coordenação de Múltiplas Visões . . . . .	25
3.7.2	Visualização e Análise Exploratória dos Dados . . . . .	26
3.8	TreeMap . . . . .	28
3.9	Como escolher que Visões utilizar? . . . . .	30
<b>4</b>	<b>Frameworks e Tecnologias para desenvolvimento web</b>	<b>31</b>
4.1	Backend . . . . .	31
4.1.1	Base de Dados . . . . .	32
4.1.2	MySQL como DML . . . . .	33
4.1.3	SQL e Queries . . . . .	34
4.1.4	LARAVEL . . . . .	34
4.2	FrontEnd . . . . .	35
4.2.1	CSS e Bootstrap . . . . .	36
4.2.2	Javascript, D3 e ZingChart . . . . .	37
4.2.3	Highlight e PRISM . . . . .	39
4.2.4	Summernote e Quill . . . . .	40
<b>5</b>	<b>Arquitetura Proposta</b>	<b>41</b>
5.1	Arquitetura de uma ferramenta de estudo de uma linguagem de programação . . . . .	41
5.2	Componente de identificação e recolha de dados sobre um repositório de código MATLAB . . . . .	42
5.3	Componente de análise sobre o repositório MATLAB . . . . .	43
5.4	Componente de representação dos dados analisados . . . . .	44
5.5	Modelo de Dados . . . . .	45
5.5.1	Decomposição de m-files em estruturas de dados . . . . .	45
5.5.2	Anotações sobre linhas de código e instâncias <i>tokens</i> . . . . .	47
5.5.3	Estruturas de dados a analisar e as suas relações no modelo de dados . . . . .	48
5.5.4	Modelo de dados como suporte ao uso do SOM . . . . .	49
<b>6</b>	<b>Implementação e Validação com Metáforas Visuais</b>	<b>53</b>
6.1	Implementação do Modelo de Dados em SQL . . . . .	53
6.2	CCCEXplorer . . . . .	55
6.2.1	Alterações realizadas na ferramenta . . . . .	55
6.2.2	Importação das estruturas de dados estudadas . . . . .	55
6.3	Representação do modelo SOM estudado na base de dados . . . . .	58
6.4	Componentes das Metáforas Visuais . . . . .	59
6.4.1	Vista de Código . . . . .	60

6.4.2	Vista Comparativa . . . . .	61
6.4.3	Módulo m-files de Neurónio . . . . .	62
6.4.4	Módulo de Anotações . . . . .	62
6.5	TreeMaps e análise do modelo relacional do modelo SOM . . . . .	65
6.5.1	Concerns - Toolboxes - M-files . . . . .	66
6.5.2	Regions - Patterns - M-files . . . . .	71
6.6	Vista do Módulo de Queries . . . . .	73
6.6.1	Queries Sequências de Tokens . . . . .	74
<b>7</b>	<b>Conclusão</b>	<b>79</b>
	<b>Bibliografia</b>	<b>81</b>



## LISTA DE FIGURAS

2.1	Esquema ilustrativo do sintoma Scattering em código <i>MATLAB</i> [24] . . . . .	7
2.2	Exemplo ilustrativo do sintoma Tangling em código <i>MATLAB</i> [12] . . . . .	7
2.3	Versão limpa da função exemplo . . . . .	8
2.4	Versão da função que contempla dois <i>concerns</i> . . . . .	8
2.5	Estrutura de classes da package main do <i>CCCExplorer</i> . . . . .	13
2.6	Estrutura de <i>packages</i> e <i>classes</i> necessárias à identificação dos <i>tokens</i> . . . . .	13
2.7	Estrutura do package metrics . . . . .	14
2.8	Classes constituintes do <i>package output_view</i> . . . . .	14
2.9	Estrutura de packages e classes . . . . .	14
3.1	Esquema ilustrativo do processo de compreensão [5] . . . . .	22
3.2	Visualização U-Matrix [26] . . . . .	28
3.3	Component Plane referente à <i>concern</i> Verificação de argumentos de funções e valores retornados [26] . . . . .	28
3.4	Component Plane referente à <i>concern</i> Verificação do tipo de dados [26] . . . . .	28
3.5	Exemplo ilustrativo de um TreeMap [7] . . . . .	29
4.1	Propriedades do sistema de colunas do Bootstrap . . . . .	36
4.2	Botões do Bootstrap . . . . .	36
4.3	Exemplo visual de um Popover . . . . .	37
4.4	Exemplo visual de um Modal . . . . .	37
4.5	Caso interessante da utilização de um componente nav versão a . . . . .	37
4.6	Caso interessante da utilização de um componente nav versão b . . . . .	37
5.1	Arquitetura Global do Sistema . . . . .	41
5.2	Esquema representativo dos processos a completar pelo <i>CCCExplorer</i> . . . . .	43
5.3	Diagrama ER da decomposição dos <i>m-files</i> . . . . .	46
5.4	Diagrama ER referente ao sistema de Anotações . . . . .	48
5.5	Diagrama ER referente ao armazenamento dos dados que alimentam as ferramentas de análise do sistema . . . . .	49
5.6	Diagrama ER relativo às várias instâncias <i>SOM</i> e estruturas inerentes ao treino do <i>SOM</i> . . . . .	50

6.1	Vista de Visualização de Código . . . . .	60
6.2	Vista Comparativa de Visualização de Código . . . . .	61
6.3	Módulo m-files do Neurónio . . . . .	62
6.4	Módulo Inserção de Anotações . . . . .	63
6.5	Módulo Visualização de Anotações . . . . .	64
6.6	Popover Token . . . . .	64
6.7	Vista TreeMap . . . . .	65
6.8	<i>TreeMap</i> representativo de todos os <i>concern</i> do repositório . . . . .	67
6.9	<i>TreeMap</i> ao nível do <i>concern</i> Data type verification . . . . .	69
6.10	<i>TreeMap</i> ao nível m-file . . . . .	70
6.11	Vista <i>TreeMap</i> com visualização do código do m-file selecionado no <i>TreeMap</i> . . . . .	70
6.12	<i>TreeMap</i> das regiões do modelo SOM estudado [26] . . . . .	72
6.13	Vista Módulo de Queries . . . . .	73



## LISTA DE TABELAS

2.1	Categorização de um <i>Token</i> . . . . .	9
2.2	Mapeamento entre <i>Concerns</i> e os <i>Tokens</i> correspondentes [32] . . . . .	11
4.1	Lista de características necessárias das ferramentas . . . . .	39
6.1	Número de toolboxes identificadas com instâncias tokens, por concern . . . .	68
6.2	Toolboxes com maior número de instâncias tokens do concern Data type verification . . . . .	68
6.3	Formato CSV utilizado na construção dos <i>TreeMaps</i> . . . . .	71
6.4	Número de neurónios pertencentes a cada região do SOM estudado [26] . .	72



## LISTAGENS

4.1	Consulta de dados em <i>SQL</i> . . . . .	34
4.2	Exemplo <i>ZingChart</i> da estrutura de dados para um <i>TreeMap</i> . . . . .	38
4.3	Exemplo <i>ZingChart</i> da estrutura de dados para um <i>HeatMap</i> . . . . .	39
6.1	Instruções <i>SQL</i> utilizadas na construção das estruturas de dados para os <i>TreeMaps</i> . . . . .	53
6.2	Código <i>SQL</i> para inserção de <i>tokens</i> na base de dados . . . . .	56
6.3	Código <i>SQL</i> para inserção de <i>toolboxes</i> na base de dados . . . . .	57
6.4	Inserção na tabela <i>mfiles</i> <i>original_mfiles</i> e <i>version_mfiles</i> . . . . .	57
6.5	Código <i>SQL</i> para inserção de linhas de código e comentário na base de dados . . . . .	57
6.6	Código <i>SQL</i> para inserção instâncias <i>token</i> na base de dados . . . . .	58
6.7	<i>Query SQL</i> que cria a relação entre os elementos constituintes do <i>TreeMap</i> . . . . .	66
6.8	Estrutura de dados <i>JSON</i> que alimenta a <i>API ZingChart</i> para representação do <i>TreeMap</i> em estudo . . . . .	66
6.9	Seleção do <i>concern</i> “Data type verification” no modelo de dados . . . . .	68
6.10	Seleção da <i>toolbox</i> “SourceForge-MATLAB-Mfiles8/spm5-2011-12-14/spm5” no modelo de dados . . . . .	69
6.11	<i>Query SQL</i> que cria a relação entre os elementos constituintes do <i>TreeMap</i> . . . . .	71
6.12	<i>Query SQL</i> que procura os <i>m-files</i> de cada região . . . . .	72
6.13	Código <i>PHP</i> que monta o <i>SELECT</i> da <i>query</i> . . . . .	74
6.14	Código <i>PHP</i> que monta o <i>FROM</i> da <i>query</i> . . . . .	75
6.15	Condição do <i>id</i> do <i>m-file</i> para cada instância <i>token</i> encontrada ser o mesmo . . . . .	76
6.16	Código <i>PHP</i> que monta o <i>WHERE</i> da <i>query</i> . . . . .	76
6.17	Código <i>PHP</i> que monta o <i>WHERE</i> da <i>query</i> . . . . .	77



## INTRODUÇÃO

A modularidade em linguagens de programação é um conceito fundamental que permite a reutilização de abstrações e implementações de funcionalidades, contribuindo para a legibilidade e compreensão do código. Na linguagem imperativa *MATLAB* verifica-se que as características modulares presentes, ficheiros m-files e funções, não são suficientes para suportar a modularidade de forma satisfatória. Além disso, verifica-se que uma parte significativa da comunidade *MATLAB* são pessoas em que a sua área principal não é a informática [12]. É recorrente em código *MATLAB* referirem-se sintomas dos limites existentes no suporte à modularidade [26] [9].

Para analisar os sintomas de falta de modularidade da linguagem *MATLAB*, o algoritmo *UbiSOM* possibilita o treino de instâncias de *Self-Organising Maps* (SOM) [37] [9]. Os mapas auto-organizados podem descrever grandes quantidades de dados multidimensionais, organizando-os topologicamente segundo a sua semelhança. O algoritmo *UbiSOM* utiliza um processo de aprendizagem automático para produzir modelos de dados que analisam métricas sobre código *MATLAB*. Esta dissertação utiliza o modelo SOM estudado no artigo “*Toward a Token-Base Approach to Concern Detection in MATLAB Sources*” [26]. O sistema *MatrixView* implementado sobre o algoritmo *UbiSOM* pode ser utilizado para possibilitar uma análise exploratória dos dados (ver secção 3.2). Nesta dissertação pretende-se estudar de que forma se podem disponibilizar ferramentas que possibilitem o acesso web e visualização dos modelos e anotações desta análise exploratória (ver secção 2.3).

Para que os modelos produzidos pelo SOM sejam compreendidos, é necessário o uso de ferramentas representativas de dados. Nesta dissertação serão utilizadas metáforas visuais e um modelo relacional. As metáforas visuais são ferramentas que representam

ideias, situações e contextos sob a forma de imagens [1]. O modelo relacional é a solução clássica que permite a gestão e acesso aos dados através de um *RDBMS* (*relational database management system*).

Para efeitos de interação com o protótipo desenvolvido no contexto desta dissertação, são considerados dois tipos de utilizadores. Consideramos utilizadores experientes que têm conhecimentos de bases de dados, de modelos SOM e da linguagem *MATLAB* – denominados por *Stephanies* – e um segundo tipo de utilizadores que são programadores da linguagem *MATLAB* – denominados por *Joes*. Utilizadores *Joes* apenas visualizam resultados enquanto que utilizadores *Stephanies* produzem também anotações e metáforas visuais para dar apoio aos *Joes* nas suas tarefas de análise.

### 1.1 Objetivos

Esta dissertação visa estudar metáforas visuais e modelos de representação de dados como forma de apoio à compreensão dos dados por parte de programadores interessados em linguagens orientadas ao processamento de matrizes. Em particular, tem-se em vista utilizadores com conhecimento da linguagem *MATLAB* porque esta é uma das linguagens onde existe maior volume de código disponível para análise e será a estudada neste trabalho.

A dissertação tem por base o trabalho descrito no artigo “*Toward a Token-Based Approach to Concern Detection in MATLAB Sources*” [26]. Esta dissertação tem por objetivo operacionalizar os modelos de análise de dados descritos neste artigo para deteção de *concerns* para programadores *MATLAB*.

Para validação desta dissertação, pretende-se desenvolver um modelo de dados que possibilite gerir toda a informação relevante e um método de visualização e análise de toda a informação relacionada. Como caso de estudo usa-se o modelo e repositório descritos e desenvolvidos num estudo anterior [26]. Os diferentes elementos propostos devem poder ser conjugados numa primeira proposta de um protótipo *web* que ilustre as possibilidades das anotações de *concerns* num repositório *MATLAB*. Estudaram-se metáforas visuais para estruturar um conjunto de métricas e propiciar a identificação de propriedades relevantes no código *MATLAB*. O protótipo *web* destina-se a estudar os *crosscutting concerns* presentes em sistemas *MATLAB*, ver secção 2.3, sobre o repositório *MATLAB* já usado em trabalhos anteriores [32] [26] [9].

Estes objetivos podem no entanto, ser decompostos num conjunto de sub-objetivos, os quais são:

- Transformar e armazenar o repositório *MATLAB* numa base de dados.

- Estudar e desenvolver um protótipo *web* para possibilitar, a análise, a visualização de resultados e a edição de anotações sobre o modelo SOM estudado no trabalho anterior [26].
- Estudar e implementar metáforas visuais para a representação de código *MATLAB* e do modelo SOM;
- Possibilitar que os utilizadores *Stephanies* possam produzir conteúdo, anotações e metáforas visuais, para que os dados resultantes do SOM possam ser mais facilmente percebidos;
- Estudar possíveis formas de melhorar a utilização dos SOMs, enquanto ferramenta de análise exploratória;
- Análise preliminar sobre a utilidade do protótipo desenvolvido e das metáforas utilizadas para representação dos dados.

## 1.2 Contribuições

Com a elaboração desta dissertação pretendo contribuir com:

- Desenvolvimento de componentes para um protótipo de aplicação *web*, nomeadamente para possibilitar, a seleção de código *MATLAB* com determinados *concerns*, a sua análise, visualização de resultados e edição de anotações sobre o código.
- Aplicação deve possibilitar um trabalho colaborativo e por isso mesmo, mais enriquecedor para os seus utilizadores;
- Implementação de componentes que auxiliam na análise exploratória de dados da linguagem *MATLAB* e a sua relação com um modelo representado num mapa auto-organizado pré-anotado.

## 1.3 Estrutura do Documento

Para além deste capítulo a dissertação é composta por:

- O segundo, terceiro e quarto capítulos destinam-se a relacionar o estado da arte com a solução pretendia e com o protótipo a implementar:
  - O segundo capítulo, Modularidade e *Concerns*, começa por caracterizar o problema e o tipo de dados que se pretende analisar, ou seja, código *MATLAB* no qual é visível a presença de *crosscutting concerns*. São apresentados os conceitos Modularidade, *Concerns*, Tokens e a abordagem na detecção de *concerns* utilizada no artigo e que é adotada nesta dissertação.

De seguida, caracteriza-se o que são métricas e quais são utilizadas como dados de análise na solução proposta. É apresentada a ferramenta utilizada para a transformação do repositório *MATLAB* em estruturas para armazenar numa base de dados.

- No capítulo três, *SOM*, *Visualização* e *Metáforas Visuais*, é introduzida uma contextualização sobre os modelos *SOM*, sobre visualizações, sobre metáforas visuais e visualização exploratória. É abordado o funcionamento do *SOM* e são abordadas algumas metáforas visuais que têm interesse em estudar para a visualização exploratória de dados e como integrar essas metáforas com a informação disponível num mapa auto-organizado. São também referidas técnicas auxiliares para *Joes* na análise das metáforas visuais.
- No capítulo quatro, *Frameworks* e *Tecnologias para desenvolvimento web*, são abordadas algumas ferramentas de desenvolvimento de aplicações *web* bem como de desenvolvimento de metáforas visuais.

O capítulo é constituído por duas secções, a primeira, secção 4.1 é referente às *frameworks* estudadas e utilizadas para modular o *backend* do protótipo enquanto que a segunda se refere às utilizadas para desenvolver o *frontend*. Apresenta a linguagem *SQL* que é um componente fundamental na construção das análises realizadas.

- O quinto capítulo, *Arquitetura Proposta*, apresenta a arquitetura proposta para a solução desenvolvida.

A primeira secção do capítulo apresenta a arquitetura global do sistema, e as seguintes apresentam de forma separada a arquitetura do sistema por detrás das três fases que constituem uma ferramenta de análise de dados, tratamento de dados, análise dos dados, secção e a representação dos resultados. A última secção deste capítulo apresenta o Modelo de Dados que serve de base para o protótipo.

- O sexto capítulo, *Implementação e Validação com Metáforas Visuais*, apresenta a arquitetura proposta. A primeira secção aborda as tabelas utilizadas na construção dos vários componentes desenvolvidos. A segunda secção aborda as alterações realizadas à ferramenta *CCCExplorer*, a terceira secção aborda a transformação do modelo *SOM* desenvolvido em trabalho anterior [26] num modelo relacional. Na quarta secção a vista de código e módulos auxiliares são apresentados, as suas implementações e modos de funcionamento são descritos. Dois tipos de *TreeMaps* são apresentados como casos de validação do protótipo e do modelo *SOM* estudado [26], onde é descrito e demonstrado de forma detalhada o funcionamento do sistema.
- O sétimo capítulo refere-se à *Conclusão*, em que é feita uma reflexão sobre os resultados obtidos tendo em conta os objetivos.



## MODULARIDADE E CONCERNS

Este capítulo tem como objetivo abordar o problema da falta de modularidade presente na linguagem *MATLAB*. O capítulo subdivide-se em 5 secções, a secção 2.1 aborda a modularidade em *MATLAB*, a secção 2.2 aborda o conceito *concern*, a secção 2.3 aborda o conceito *crosscutting concerns* na sua generalidade e especificamente na linguagem *MATLAB*, a secção 2.4 aborda o tipo de abordagens para mineração de dados e especificamente a que se utilizou nesta dissertação e a secção 2.5 aborda a ferramenta utilizada nesta dissertação de análise léxica da linguagem *MATLAB* que extrai métricas sobre a mesma.

### 2.1 Modularidade

A modularização pode ocorrer de forma abstrata, no desenho, ou concretamente, na implementação [34]. A modularidade pode ser mais geral, tal como a estrutura de um módulo de uma linguagem de programação, ou então muito específica, tal como a modularidade necessária numa estrutura *MVC* para sistemas interativos.

Existem normalmente 3 classes de razões para a utilização de modularidade em sistemas de software:

- **Controlo Intelectual:** Os programadores normalmente modularizam conceitos para que estes possam ser usados de forma isolada para facilitar a procura de informação relacionada com aspetos específicos do sistema e para que o entendimento de uma parte do mesmo não seja complicada por detalhes de outras partes. São utilizadas arquiteturas tradicionais com tipos específicos de módulos de forma a que se possa reutilizar conhecimento e de forma a tornar o sistema perceptível por outros.

- Segmentação do Trabalho: Modularizam-se conceitos de forma a que se possam separar responsabilidades, corresponder tarefas a determinadas habilidades, e fatorizar tarefas que correspondem a corpos de conhecimento.
- Evolução: Modularizam-se conceitos para que seja necessário fazer alterações apenas num só local, para que possamos substituir módulos antigos por novos, para que se possa compor elementos predefinidos e para harmonizar a evolução do sistema.

As unidades modulares (módulos) presentes na linguagem *MATLAB* são os m-files e funções. Um m-file é composto por múltiplas linhas sequenciais de comandos *MATLAB* e chamadas de funções [40].

A modularidade é muitas vezes utilizada de forma limitada pelos utilizadores *MATLAB*. Os programas *MATLAB* podem ser disponibilizados e partilhados através de pacotes (denominados *toolboxes*). Estas *toolboxes* são normalmente constituídas por conjuntos de m-files.

## 2.2 Concerns

Na ciência da computação, um *concern* [11] é um conjunto particular de informações que afeta o código de um programa de computador [17]. Em modularidade um *concern* é qualquer abstração, conceito, ou conjunto consistente de responsabilidades que gostaríamos de localizar no seu próprio módulo [32] [17].

Um algoritmo por norma em programação ou uma instrução tem um tipo de funcionalidade associada, como por exemplo, alocação de memória, verificação de tipos de dados, mensagens da linha de comandos, entre outras. Um *concern*, no âmbito do problema, pode ser visto como sendo um conjunto de características representativas do objetivo de usar determinados algoritmos, exemplo, o *concern* de alocação de memória engloba “qualquer” tipo de instrução e/ou algoritmo que tenha como finalidade a alocação de memória. Idealmente, cada *concern* mapeia para um determinado módulo, contribuindo para uma melhor modularidade do sistema.

## 2.3 CrossCutting Concerns

*CrossCutting* é um termo utilizado para descrever *concerns* em que o código destes atravessa a estrutura modular de um sistema de *software*, sendo estes denominados normalmente pelo acrónimo CCC [32]. A identificação destes *crosscutting concerns* comprova que as partes principais do sistema encontram-se “poluídas” [32].

No passado, foram identificados dois tipos de sintomas que resultam da presença de *crosscutting concerns*, sendo eles:

- **Scattering(Dispersão)** representado na Figura 2.1, onde é visível que a instrução `moveBy(int, int)` se encontra em mais do que um módulo (*Figure* e *Line*), característica representativa do *crosscutting concern* em questão, código repetido e espalhado pelo programa [32];
- **Tangling(Emaranhado)** representado na Figura 2.2, em que podemos observar, recorrendo à Tabela 2.2, a característica principal representativa do *crosscutting concern* em questão, existe o uso de diferentes *concerns* no “mesmo contexto de programação”. O código que pertence a um determinado *concern* está misturado com o código que pertence a um ou mais tipos diferentes de *concerns* [32].

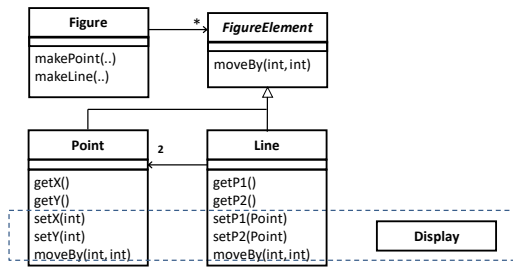


Figura 2.1: Esquema ilustrativo do sintoma Scattering em código MATLAB [24]

```
public class Point implements FigureElement {
    private int _x, _y;
    private Display _display;

    public Point(int x, int y) {
        _x = x;
        _y = y;
    }

    public Point(int x, int y, Display display) {
        this(x, y);
        setDisplay(display);
    }

    public void setX(int x) {
        _x = x;
        _display.update(this);
    }

    public void setY(int y) {
        _y = y;
        _display.update(this);
    }

    public void setDisplay(Display display) {
        _display = display;
    }

    public void moveBy(int dx, int dy) {
        _x += dx;
        _y += dy;
        _display.update(this);
    }
}
```

Figura 2.2: Exemplo ilustrativo do sintoma Tangling em código MATLAB [12]

Não é garantido que sejam diretamente transportáveis trabalhos sobre a modularidade das linguagens, nomeadamente sobre linguagens Orientadas a Objetos e sobre a linguagem C, para o estudo da modularidade de sistemas MATLAB. Tal facto, deve-se aos sistemas MATLAB serem muito diferentes em relação aos sistemas orientados a objetos, tanto nas características da linguagem como nos usos da mesma [24]. Por isso mesmo, existiu a necessidade de uma abordagem dirigida ao estudo da modularidade dos sistemas MATLAB.

### 2.3.1 CCC em MATLAB

O par de Figuras 2.3 e 2.4 representa um dos sintomas causado pela falta de suporte à modularidade, *code tangling*. Como referido na secção 2.3, *code tangling* pode ser identificado quando uma função individual contém mais do que um *concern* presente.

```
function [y] = dft(x)
y=zeros(size(x));
N=length(x);
t=(0:N-1)/N;
for k=1:N
    y(k) = sum(x.*exp(-j*2*pi*(k-1)*t));
End
```

Figura 2.3: Versão limpa da função exemplo

```
function [y] = dft_specialized(x)
y=zeros(size(x));
N=length(x);
t=(0:N-1)/N;
quant1=quantizer('fixed','floor','wrap',[18 16]);
t=quantize(quant1,t);
quant2=quantizer('fixed','floor','wrap',[23 20]);
pi_fix = quantize(quant2,pi);
quant3=quantizer('fixed','floor','wrap',[20 8]);
quant4=quantizer('fixed','floor','wrap',[23 10]);
quant5=quantizer('fixed','floor','wrap',[24 10]);
quant6=quantizer('fixed','floor','wrap',[26 12]);
quant7=quantizer('fixed','floor','wrap',[28 14]);
quant8=quantizer('fixed','floor','wrap',[32 16]);
for k=1:N
    v1 = quantize(quant3,(k-1)*t);
    v2 = quantize(quant4,pi_fix*v1);
    v3 = quantize(quant5,-j*2*v2);
    v4 = quantize(quant6,exp(v3));
    v5 = quantize(quant7,x.*v4);
    y(k) = quantize(quant8,sum(v5));
end
```

Figura 2.4: Versão da função que contempla dois *concerns*

Nas Figuras 2.3 e 2.4 estão representadas duas implementações diferentes da função *Discrete Fourier Transform*, que é normalmente usada no processamento de sinais [12].

A Figura 2.3 representa uma versão limpa da função com apenas um *concern* presente. A Figura 2.4 representa uma versão mais complexa da função e onde existe a presença de mais do que um *concern*, o *Data type Specialization*. Este *concern* está representado pelas palavras com um fundo colorido, *quantize* e *quantizer*. Por isso mesmo, a Figura 2.3 é um ótimo exemplo de *code tangling*.

## 2.4 Abordagem para a detecção de concerns em MATLAB

### 2.4.1 Mineração de Aspectos

A tarefa de indicar e localizar as CCCs em código existente é designada como mineração de aspectos [16]. Pesquisas típicas sobre mineração de aspectos abrangem o desenvolvimento de métodos promissores a serem extraídos dos seus próprios módulos de aspecto. Uma vez concluída a identificação, um processo de refatoração pode ser realizado para a extração de aspectos que produzam uma versão orientada a aspectos do sistema original alvo [25].

A pesquisa sobre a mineração de aspectos é amplamente focada no paradigma OO, especialmente no caso específico da linguagem de programação C [4]. Java é o representante OO, mais comum [16]. Verifica-se que o código *MATLAB* coloca desafios diferentes, possivelmente mais desafiantes para tarefas de mineração de aspectos [24].

O *MATLAB* não suporta o equivalente a *APIs* em sistemas OO. A maioria dos elementos do *MATLAB* disponíveis para análise são detalhes minuciosos das implementações de funções, que acabam por ser ocultadas das *APIs*, como nomes de variáveis locais, estruturas de controle e nomes de funções de chamada.

A presença de *concerns* secundários no *MATLAB* geralmente resulta em código modificado cujos detalhes dependem da lógica primária original e do *concern* secundário em questão.

Um tipo de técnica de mineração de aspecto são as técnicas de detecção de clones que visam encontrar código duplicado, que pode ter sido adaptado ligeiramente do original.

As técnicas de detecção de clones baseadas em *tokens* são sobre a realização de uma tokenização do código-fonte e, subsequentemente, usam os *tokens* como base para a detecção de clones [4].

### 2.4.2 Token e Word Token

Nesta sub-secção são abordadas as noções de *token* e um tipo particular de *token*, *token palavra* (*word token*). O *token* é a estrutura base de suporte à abordagem de detecção de *concerns* considerada neste trabalho.

**Token:** Em programação, um *token* é um elemento singular de uma linguagem de programação. I.e. As palavras reservadas '*new*' e '*function*' são *tokens* da linguagem *JavaScript*. Operadores como  $+$ ,  $-$ ,  $*$  e  $/$ , são *tokens* de quase todas as linguagens de programação [39].

**Word token:** Os *word token* incluem as palavras reservadas, ver linha *keyword* da Tabela 2.1, e nomes ou identificadores, que consistem em quaisquer palavras que não são palavras reservadas.

Um subconjunto dos *word tokens* serão utilizados como estruturas de identificação de *concerns* nesta dissertação.

### 2.4.3 Mineração de concerns baseada em tokens

A abordagem utilizada neste trabalho baseia-se na análise de métricas que capturam informações sobre os *concerns* presentes em um ou mais módulos. Esta abordagem difere das métricas tradicionais de *modularidade* na medida em que estas últimas limitam-se a capturar informações respeitantes aos módulos já existentes.

nonkeyword	COM, ID, STR, NUM
keyword	break, case, catch, classdef, continue, else, elseif, end, for, function, global, if, otherwise, parfor, persistent, return, spmd, switch, try, whyle, nargin, etc
operadores lógicos	&, &&,  ,   , , =, ==, =, <, <=, >, >=
operadores aritméticos	+, -, *, /, ""
símbolos	(, ), [, ], {, }, :=, ;, ,, ., :, @

Tabela 2.1: Categorização de um *Token*

As métricas sobre *concerns* são particularmente promissoras para apoiar tarefas de mineração dos ditos. Nas métricas consideradas neste trabalho, a principal unidade de informação são os *tokens*, ou seja, elementos lexicais extraídos de um código por meio de uma ferramenta de análise léxica. A abordagem defende que cada *token* deve se relacionar no máximo com um *concern*.

Uma abordagem apresentada em trabalho anterior [24] defende que grupos específicos de *tokens* podem ser associados a *concerns* específicos, para efeitos de mineração dos ditos, que através do estudo dos padrões de presença de certos pares de *tokens* deve ser possível prever a ocorrência dos *concerns* a que estes tokens pertencem. Este conceito é o núcleo da abordagem baseada em *tokens* [26] [9].

Nesta abordagem, não é realista esperar que todos os *tokens* possíveis constituam bons indicadores no estudo dos *concerns* em *MATLAB*. Os *tokens* considerados no estudo dos *concerns* são palavras da linguagem *MATLAB* [32]. Na Tabela 2.2 estão representados os *concerns* e os *tokens* de cada *concern*, que até agora foram considerados no âmbito do problema em questão. A classificação presente dos *tokens* na Tabela 2.2 por *concerns* surgiu do trabalho desenvolvido no contexto da dissertação de Bruno Palma [32], tendo sido também objeto de publicação [26] [9].

Os *tokens* classificados não incluem nomes de variáveis ou quaisquer outro tipo de expressões. A sua consideração teria um impacto negativo no estudo dos *concerns*. Nomes de variáveis são atribuídos pelos programadores e esta atribuição embora possa seguir um certo critério (normalmente pessoal), não garante que o nome atribuído a uma variável está ou não relacionado com o contexto em que é utilizado, não devendo assim estes tipos de *tokens* serem enquadrados no estudo.

A identificação das ocorrências de *tokens* no código *MATLAB* é realizada pela ferramenta *CCCEXplorer*, ver secção 2.5, em que a sua função principal e de raiz, é identificar a presença de *tokens* em m-files e posteriormente gerar um conjunto de métricas que possibilitam o estudo dos *crosscutting concerns* na linguagem *MATLAB*.

#### 2.4.4 Métricas

Métricas são propriedades que permitem o estudo e análise de um tipo de dados. A métrica estudada e utilizada nesta dissertação tem como objetivo o estudo de *crosscutting concerns*. Nesta dissertação continua-se o estudo sobre a métrica *Token Density* [26] Esta métrica foi anteriormente utilizada para facilitar a identificação de *concerns* em código *MATLAB* e permitir a análise dos mesmos e das relações existentes entre si.

Como já referido as métricas geradas a partir dos m-files têm como objetivo o estudo dos *crosscutting concerns* presentes neste tipo de ficheiros. As métricas produzidas por

## 2.4. ABORDAGEM PARA A DETECÇÃO DE CONCERNS EM MATLAB

1. Verification of function arguments and return values	nargchk, nargin, nargout, nargoutchk, varargin, varargout
2. Data type specialization	fi, fimath, int16, int32, int64, int8, quantize, quantizer, sfi, single, ufi, uint16, uint32, uint64, uint8, double
3. Data type verification	cast, class, intmax, intmin, isa, isboolean, iscell, ischar, iscolumn, isempty, isfi, isfield, isfimath, isfixed, isfloat, isinf, isinfinite, isinteger, islogical, isnan, isnumeric, isobject, isquantizer, isreal, isrow, isscalar, isstr, isstruct, isvector, length, ndims, numel, range, realmax, realmin, size, typecast, wordlength
4. Dynamic properties	eval, evalc, evalin, inline, feval
5. Console messages	annotation, assert, disp, display, error, last, lastwarn
6. Printing	orient, print, printdlg, printopt
7. Visualization	aaxes, axis, box, cla, clabel, clf, close, datacursormode, datetick, errorbar, figure, figurepalette, fplot, frame2im, gca, gcbf, gcbo, gco, getframe, gplot, grid, gtext, hist, histogram, hold, im2frame, image, imfinfo, imformats, imread, imwrite, ishold, legend, line, loglog, mesh, meshgrid, movie, newplot, pan, plot, plot3, plotbrowser, plottedit, plottools, plotyy, polar, propertyeditor, rectangle, reset, rgbplot, rotate, rotate3d, scatter, semilogx, semilogy, set, showplottool, subplot, surf, textlabel, text, title, VideoReader, VideoWriter, xlabel, ylabel, zlabel, zoom
8. File I/O	diary, fgetl, fgets, fileformats, fopen, fprintf, fread, fscanf, fwrite, hgload, hgsave, load, save, saveas, uisave
9. System	addtodate, ans, ba, bafter, batch, break, calendar, calllib, clear, clearvars, clock, cputime, date, dbcont, dbmex, dbquit, dbstop, ebreak, echo, etime, exist, inmem, input, inputname, inputParser, isglobal, iskeyword, isvarname, libfunctionsview, libisloaded, loadlibrary, memory, mex, mexext, mfilename, mislocked, mlock, munlock, namelengthmax, nanbreak, next, now, onCleanup, pack, pause, pcode, rbreak, rehash, run, slist, spmd, start, startat, step, stop, symvar, systems, tbreak, tic, timerfind, timerfindall, toc, unloadlibrary, wait, weekday, where, who, whos, xbreak, zcbreak
10. Memory allocation/deallocation	delete, global, ones, persistent, zeros
11. Parallelization	cancel, codistributed, codistributor, createParallelJob, createTask, defaultParallelConfig, demote, destroy, detupForParallelExecution, dfeval, dfevalasync, distributed, gather, gcat, gop, gplus, gpuArray, gpuDevice, gpuDeviceCount, importParallelConfig, isreplicated, jobStartup, labBarrier, labBroadcast, labindex, labProbe, matlabpool, mpiLibConf, mpiprofile, mpiSettings, numlabs, parfor, pctconfig, pctRunDeployedCleanup, pctRunOnAll, pload, pmode, poolStartup, promote, psave, redistribute, resume, sparse, submit, subsasgn, subsref, taskFinish, taskStartup

Tabela 2.2: Mapeamento entre *Concerns* e os *Tokens* correspondentes [32]

parte do *CCCExplorer* para o estudo das *crosscutting concerns* são:

- **LoC - Lines of Code**, para cada m-file o número de linhas não vazias e não comentadas são contabilizadas [32];
- **Token Density** duma função ou m-file, é a divisão entre o número de ocorrências de um conjunto de tokens associados a determinado concern pelo LoC dessa função ou m-file, ou seja, para um determinado *concern* e m-file, é a normalização da contagem dos *tokens* associados ao *concern*, por meio do *LoC* desse m-file. [32].

Os valores da métrica *Token Density* são utilizados para o treino de modelos SOM. Valores elevados da métrica *Token Density* para mais do que um *concern* num m-file pode ser um fator indicativo que um *concern* está presente devido à presença de outro *concern* e/ou vice-versa. Fundamentalmente, a presença de ocorrências *tokens* pertencentes a um *concern* e igual repetição de ocorrências *tokens* de um outro *concern*, pode indicar que existe uma relação entre os *concerns* e por isso mesmo justificar a presença simultânea ou não num m-file. Contudo, a probabilidade não é pequena de num m-file com muitas linhas de código existir mais do que um *concern* presente tendo em conta a falta de modularidade na linguagem *MATLAB*. Mais, existe a possibilidade de que os *concerns* apareçam em funções diferentes e por isso mesmo o fenómeno *crosscutting concerns* pode não se verificar na medida em que os diferentes *concerns* ocorrem em contextos de programação diferentes. Dessa forma é necessário criar mais métricas e visualizações que nos ajudem a obter respostas a estas dúvidas.

## 2.5 CCCExplorer: ferramenta de extracção de métricas

O *CCCExplorer* é uma ferramenta java desenvolvida no âmbito de trabalhos anteriores [24] [26]. Esta ferramenta assenta fundamentalmente num analisador léxico de código *MATLAB* que por meio de produção das métricas *LoC* e *Token Density* permite o estudo das *crosscutting concerns*. A métrica *LoC* é calculada pelo *CCCExplorer* através da contagem de linhas de código de um m-file em que as linhas de comentário não são contempladas. Durante a análise léxica do código *MATLAB* o *CCCExplorer* identifica as ocorrências *tokens* categorizadas na Tabela 2.2 e conjuntamente com a métrica *LoC* produz a métrica *Token Density*.

Na Figura 2.5 é possível visualizar que existem várias *mains* no *CCCExplorer*. Cada uma possui uma finalidade diferente. A *MainGenBlocks.java* é a *main* a utilizar para que seja produzido um ficheiro sobre as ocorrências *tokens* presentes por bloco no m-file. A *MainbigCSV.java* cria um ficheiro com os mesmos dados que a *main* anteriormente referida, no entanto, um ficheiro CSV é produzido com estas métricas. As outras *mains* são classes orientadas para funcionalidades específicas e para produção



de outras métricas que ao longo do trabalho desta dissertação não foram utilizadas e por isso mesmo não são abordadas.

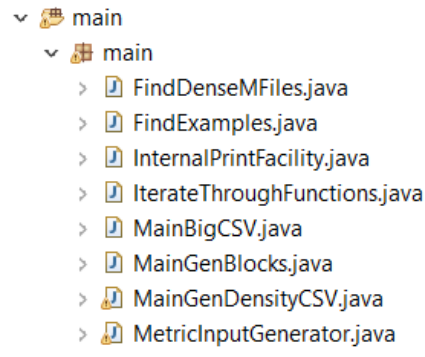


Figura 2.5: Estrutura de classes da package main do *CCCExplorer*

Numa primeira iteração, o *CCCExplorer* identifica os vários elementos léxicos presentes nos ficheiros de código *MATLAB*. A *package token\_classifier* contém as classes necessárias à implementação das funcionalidades necessárias ao analisador léxico da linguagem *MATLAB*.

Numa segunda iteração, a ferramenta identifica as ocorrências *tokens* presentes nos códigos e produz métricas referentes a estes, sendo algumas delas o número de ocorrências *tokens* presentes em cada *m-file*, o *LoC* e a *Token Density*.

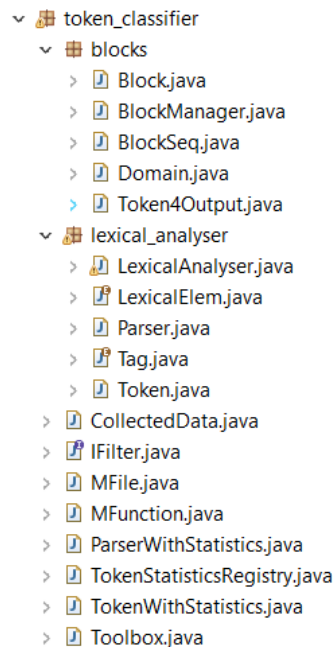


Figura 2.6: Estrutura de *packages* e *classes* necessárias à identificação dos *tokens*

Uma breve descrição das classes presentes no package *token\_classifier* e dos próprios *packages* contidos no package referido é apresentada:

- *blocks*: *Package* composto pelas classes que mantêm a informação dos blocos presentes nos *m-files* (*Block.java*, *BlockManager.java*; *BlockSeq.java*). A classe *Token4Output.java* imprime para um ficheiro de *output* a informação de cada ocorrência *token*, mais propriamente, a *TAG* que identifica o tipo de *token* da instância em termos léxicos e o nome do *token*.
- *lexical\_analyser*: *Package* possui as classes que identificam os vários elementos constituintes em termos lexicais. A classe *LexicalAnalyser.java* é a classe que analisa o *m-file* de linha a linha e de ocorrência *token* a ocorrência *token*.
- As restantes classes que estão na raiz do *package token\_classifier* são classes auxiliares à ferramenta para retenção de informação referente tanto ao *m-file* bem como aos seus elementos constituintes.

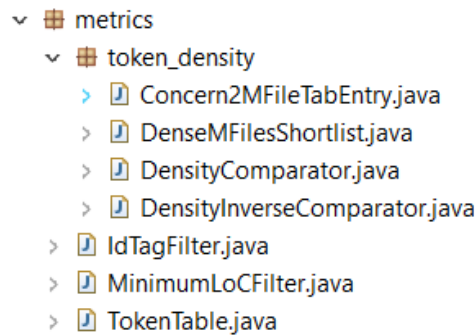


Figura 2.7: Estrutura do package metrics

O *package metrics* possui classes para produção das variadas métricas implementadas no CCCExplorer sobre o código *MATLAB*.

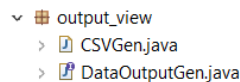
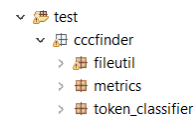
Figura 2.8: Classes constituintes do *package output\_view*

Figura 2.9: Estrutura de packages e classes

O *package output* consiste em duas classes que geram tipos diferentes de *output* dos dados gerados pelo CCCExplorer, seja em CSV como em ficheiros de texto estruturados para representação das métricas identificadas por cada bloco.

O CCCExplorer é composto ainda por um *package* denominado *test* que contém testes unitários para testar e validar as várias funcionalidades da ferramenta.

As métricas abordadas e produzidas pelo CCCExplorer abrem caminho para análises sobre as *crosscutting concerns* em código *MATLAB* contudo, tendo em consideração o

repositório com um número elevado de m-files, o uso exclusivo destas métricas tornam-se ineficazes para lidar com um número mais elevado de *concerns* (mais do que dois ou três).

Dessa forma, é necessário recorrer a uma ferramenta capaz de providenciar um panorama de todos os *concerns* em estudo e das várias relações implicadas nos dados [26].



## SOM, VISUALIZAÇÃO E METÁFORAS VISUAIS

Este capítulo tem como objetivo contextualizar os mapas SOM como ferramenta de análise de grandes quantidades de dados multidimensionais e abordar a importância da visualização e das ferramentas utilizadas na representação das conclusões provenientes de análises. Tal como referido no início do capítulo 1, nesta dissertação vai se estudar o modelo SOM abordado no artigo [26] e por isso mesmo o SOM é apenas contextualizado como ferramenta de anotação sobre o código *MATLAB*. As secções 3.1, 3.1.1, 3.1.2, introduzem o conceito SOM, fazendo a ponte com a visualização de resultados. A representação e visualização dos dados é importante na utilidade dos dados para os utilizadores do protótipo. Na secção 3.4 a visualização de dados e as metáforas visuais 3.5 como representações de dados são contextualizadas no problema, secção 3.7.2.

### 3.1 Mapas Auto Organizados

Um SOM, acrónimo para o termo em língua inglesa *Self Organizing Map*, é denominado como “Mapas Auto-organizados”. O SOM surge como uma solução para a representação das relações existentes num conjunto de dados. Estes mapas oferecem vantagens na visualização e na exploração de grandes conjuntos de dados multi-dimensionais quando comparadas com as ferramentas tradicionais.

O *UbiSOM* foi a ferramenta de análise e de visualização utilizada sobre os dados produzidos no artigo [26]. Esta ferramenta visa a ser utilizada por utilizadores *Stephanies* para extrair metáforas visuais sobre os dados produzidos pelos modelos treinados.

O SOM é usado para processar de forma não supervisionada conjuntos de dados multi-dimensionais. O algoritmo de aprendizagem é aplicado sobre cada padrão de dados de

entrada. No final, uma representação dos dados recebidos é exibida na forma de um mapa, normalmente duas dimensões. [32] Esta característica, faz com que o SOM seja distinto de outros algoritmos de agrupamento. De facto, o SOM possibilita a geração de distintas regiões no seu mapa, nomeadamente quando são detectadas diferentes relações entre as diferentes métricas analisadas. No caso em estudo [25], estas diferentes regiões do mapa estão associadas à presença simultânea de vários *concerns* no código.

Objetivo principal desta ferramenta é providenciar informação de alto-nível proveniente de grandes conjuntos de dados de baixo-nível. A característica do SOM que fez com que este fosse utilizado em trabalhos anteriores [26] foi a capacidade de produzir uma análise exploratória sobre os dados e consequentemente produzir representações dos resultados gerados.

### 3.1.1 Fase de Treino dos Mapas

Embora área de foco da dissertação não seja a análise exploratória mas sim a representação dos dados, selecção e visualização dos dados provenientes desta análise, é necessário alguma contextualização sobre a mesma, neste caso o SOM.

Durante a fase de treino dos mapas, um algoritmo de aprendizagem não supervisionado, como já referido no início desta secção, é aplicado ao conjunto de dados de entrada. Diz-se que a aprendizagem é não supervisionada pois o algoritmo por detrás é utilizado para extrair inferências de conjuntos de dados não rotulados [22]. O SOM não requer também qualquer intervenção humana durante a fase de treino e não precisa de informação prévia sobre os dados de entrada [26].

O algoritmo SOM é diferente dos típicos algoritmos de agrupamento (*clustering*) na medida em que ao aglomerar os dados deteta *clusters* num mapa, assim através da noção de vizinhança no mapa podem ser encontradas regiões (grupos de grupos) que contêm conjuntos de dados com características semelhantes, representando no nosso caso um conjunto de *m-files* semelhantes. No entanto, ao contrário de outros algoritmos de agrupamento, o SOM reduz significativamente o risco de agrupar demasiado os dados, ou seja, permite a formação de *mini-clusters* que retêm propriedades muito próprias [26].

A formação de clusters é uma forma de anotação que o SOM realiza sobre os dados de entrada.

Em trabalho anterior [26], utilizou-se o *UbiSOM* que é uma extensão do SOM como forma de treino de mapas dos dados *Token Density*. A partir deste momento e até ao fim da dissertação, referências ao algoritmo SOM pretendem referir o algoritmo *UbiSOM*. O *UbiSOM* possui duas características vantajosas para o projeto que enquadra este trabalho, sendo elas:

- a identificação de variações significativas nos dados de entrada, acomodando dessa forma o conceito *streaming*;
- uma melhor parametrização, ou seja, aquando da interação com os mapas, seleção de um subconjunto de estudo, ou na escolha de métricas a analisar, o *UbiSOM* consegue acomodar essas alterações sem que tenha recomeçar o processo de treino desde o início.

A capacidade de suportar a interação com o mapa deve-se ao Ubi-SOM na fase de treino consistir numa máquina de dois estados, organização e aprendizagem. A alternância entre estes dois estados durante a fase de treino concede ao mesmo uma propriedade de plasticidade.

O SOM processa a *Token Density* de cada m-file do repositório. Cada padrão processado é um vetor  $X = [x_{1k}, x_{2k}, \dots, x_{Nk}]$ , onde cada  $x_{ik}$  é calculado por uma fórmula. Este tipo de padrões são representações chave na solução proposta, ver sub-secção 5.5.3.

O algoritmo SOM consiste em várias etapas ao longo de várias iterações que atualizam o modelo SOM a obter. Neste trabalho, um modelo SOM é um mapa bidimensional (com  $K=20 \times 40$  posições), em que cada posição (célula) nesse mapa é um neurónio [26]. Um neurónio é a estrutura mínima que constitui uma rede neuronal, cada peso associado a um neurónio no SOM é inicializado com um valor aleatório. De seguida, um vetor  $X$  é escolhido aleatoriamente do conjunto de dados de entrada. A distância *Euclidiana* é calculada em relação a todos os vetores de pesos presentes no SOM de forma a calcular a *BMU* (*Best Match Unit*).

- A *BMU* é o neurónio com o vetor de pesos mais próximo do vetor  $X$  de entrada. Esta é o neurónio com a menor distância *Euclidiana* entre o seu vetor de pesos e o vetor  $X$  de entrada. Esta distância mínima é denominada por erro de quantização, propriedade utilizada para avaliar a performance do modelo.

Após encontrada a *BMU* os vetores de pesos do mapa são atualizados de forma a aproximar no espaço a *BMU* o mais possível do vetor de entrada. A vizinhança dos neurónios da *BMU* são ajustados de igual forma. O processo repete-se até que o mapa seja considerado estável. Para que o mapa seja considerado estável, por exemplo o utilizador pode considerar que o erro de quantização é já suficientemente pequeno, ou o número de iterações para que o SOM foi configurado iterar seja atingido.

### 3.1.2 Interpretação do SOM

Os SOM estabelecem uma projeção topológica ordenada dos dados de entrada com um número fixo  $K$  de neurónios, os quais possuem relações de semelhança com a sua vizinhança.

O SOM tradicional, normalmente é constituído por uma grelha retangular formada por estruturas ordenadas denominadas por *nós* ou *neurónios*. Os neurónios resultam do mapeamento dos dados de entrada dos SOM, descrito na sub-secção anterior 3.1.1. Possuem um vetor como já referido e uma posição na representação. O mapeamento dos neurónios é feito através das relações existentes entre os dados.

A disposição dos neurónios no mapa tem uma ordem associada. Diferentes neurónios com vetores de valores semelhantes representam neurónios em que o código *MATLAB* que os codifica apresenta relações entre eles. Quanto menor for a semelhança entre os códigos *MATLAB*, maior será a distancia entre estes os neurónios que os contêm. Assim, a projeção dos dados, representa de uma forma muito fiel as relações presentes entre estes [26].

Considerando o nosso caso de estudo e tendo em conta o referido, na representação do SOM, m-files com presença de *concerns* semelhantes são associados a uma unidade/-posição do mapa, ou seja, neurónio. A formação de regiões por parte do SOM deve-se aos valores da métrica *Token Density* dos m-files associados a neurónios pertencentes de uma região possuírem serem semelhantes, ver secção 2.4.4. Dessa forma, torna-se possível a análise exploratória para detetar correlações entre *concerns*, através de padrões representados pela disposição dos neurónios.

Mais à frente voltaremos à interpretação dos SOM, onde consideramos as visualizações produzidas por este, mas agora é importante contextualizar o conceito visualização.

## 3.2 Atributos Reais e Atributos Visuais

Na representação gráfica de atributos reais é necessário fazer o mapeamento destes para atributos visuais. Um atributo visual visa ser uma representação de um atributo real. Consideremos o exemplo seguinte, representar o tamanho em termos de espaço ocupado na memória de um *software* é um atributo que real que decidimos representar graficamente. Necessitamos primeiramente de definir como esta propriedade vai ser representada graficamente, devido a isso mesmo, um mapeamento é feito e podemos definir, como exemplo, que esta propriedade será representada através de um conjunto de cores. Assim, acabamos de fazer um mapeamento de um atributo real para um atributo visual e podemos agora fazer uma representação gráfica desta propriedade do elemento de *software*.

Na base do nosso problema, os atributos reais a representar serão principalmente as *Token Density*. A forma como iremos representar visualmente esta informação, dependerá das metáforas visuais utilizadas. Dessa forma, é necessário analisar quais as ferramentas que mais se adequam ao problema em estudo.



### 3.3 SourceMiner

O *SourceMiner* é uma aplicação do tipo AIMV (*Ambiente Interativo baseado em Múltiplas Visões*) utilizada na representação, através de metáforas visuais, de informações relevantes de uma aplicação *Java* [5]. A aplicação foi desenvolvida como um *plug-in* do *Eclipse* com a finalidade de apoio a atividades de compreensão de *software* [36].

O *SourceMiner* foi desenvolvido e posteriormente estendido de forma a suportar a criação, de filtros, de ferramentas, visões de filtragem e de visões. A aplicação disponibiliza uma visão de, grafos, polimétrica, *TreeMap*, acoplamento e uma visualização tabular [36].

### 3.4 Visualização

A visualização é um importante meio de compreensão e é fundamental para apoiar a construção de um modelo mental ou uma imagem mental a respeito de alguma representação visual [38]. A visualização é o processo no qual dados são transformados em imagens, as quais são posteriormente interpretadas permitindo a descoberta de novas informações.

A visualização deve permitir que os seres humanos utilizem o raciocínio perceptivo (baseado na percepção visual) ao invés de utilizar o raciocínio cognitivo na detecção de padrões. Assim deve-se utilizar simbologias familiares ao ser humano pois automaticamente estas permitem a associação dos conceitos correspondentes. Desta forma a detecção dos padrões presentes nas visualizações torna-se mais fácil. Com a redução do esforço cognitivo na detecção dos padrões torna-se possível que as capacidades cognitivas sejam utilizadas o máximo possível na interpretação dos padrões [27].

Processos complementares à visualização devem ser utilizados de forma a facilitar o processo de interpretação por parte dos utilizadores. A interatividade é um processo complementar importante. Os atuais sistemas de visualização de informação disponibilizam ferramentas de interatividade aumentando a efetividade da procura de informação. [42]

O processo de visualização é uma etapa no processo de compreensão. O processo de compreensão, como é possível ser observado na Figura 3.1 tem como início um conjunto de dados e tem como fim atingir um conhecimento aprofundado a respeito dos dados [5].

A visualização encontra-se entre os dados e a informação. Esta é um mecanismo que proporciona métodos e técnicas para organizar e representar os dados para que deles seja obtida informação.

Quando a informação é integrada à experiência surge o conhecimento. Através da experiência é possível adquirir conhecimento a partir da interpretação dos factos que

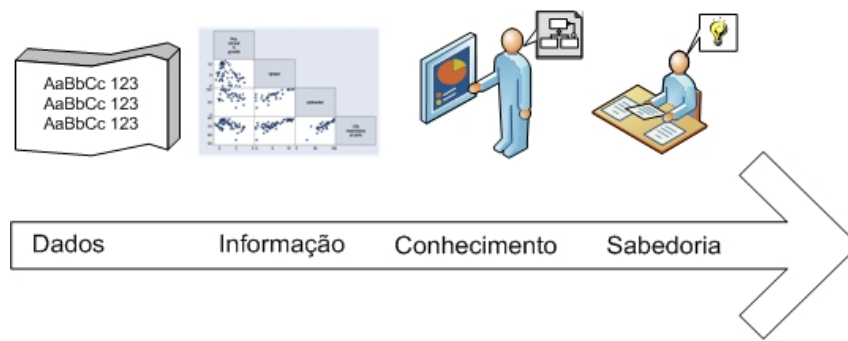


Figura 3.1: Esquema ilustrativo do processo de compreensão [5]

nos rodeiam. A sabedoria é o nível mais avançado do processo de compreensão. Com o uso desta, é possível fazer-se um julgamento qualificado sobre os dados. A sabedoria é a consequência da habilidade de estudar e interpretar o próprio conhecimento, no entanto, ao contrário do conhecimento a sabedoria não é transmissível.

Segundo *Keim*, os objetivos da visualização de informação dividem-se em três tipos:

- **Análise Exploratória:** Não há uma hipótese a respeito dos dados; o conhecimento é descoberto através de uma busca interativa.
- **Análise Confirmativa:** Há uma hipótese formulada; o objetivo é conhecido e o conhecimento é confirmado ou rejeitado durante a busca do objetivo.
- **Apresentação:** Factos e informações são previamente conhecidos e apresentados a outros com o auxílio de uma ferramenta de visualização.

Neste trabalho o nosso problema enquadra-se fundamentalmente na Análise Exploratória, pelo que será a única que será contextualizada e abordada. No entanto, uma Análise Confirmativa também será realizada na fase de validação da solução desenvolvida.

### 3.5 Metáforas e Visualização Exploratória

A análise exploratória geralmente é aplicada em dados multi-dimensionais nos quais existe um número considerável de variáveis (ou atributos). Tendo em conta a multi-dimensionalidade dos dados torna-se difícil inferir conclusões sobre a totalidade dos mesmos. Contudo, existem ferramentas que visam facilitar a análise dos mesmos, das quais as metáforas visuais fazem parte. A representação visual segundo metáforas visuais permite a descoberta de padrões entre os dados permitindo assim inferir informação sobre os dados enquanto um conjunto ao invés da análise ser feita sobre os dados de uma forma isolada. [5, 13]

### 3.5.1 Categorização das Metáforas Visuais

As metáforas visuais encontram-se divididas em categorias as quais são apresentadas de seguida, (apenas as categorias mais relevantes tendo em conta o problema serão apresentadas):

- **Metáforas Orientadas a Pixel:** Os dados são primeiramente mapeados em pixeis e consequentemente são coloridos, ou seja, são atribuídos aos dados cores dependendo do seu valor. A disposição espacial dos pixeis que representam os dados é suficiente para representar os agrupamentos de elementos com valores semelhantes. [5]

Um *HeatMap*, exemplos abordados secção 3.7.2, é uma representação gráfica dos dados em que os valores individuais contidos numa matriz são representados por um conjunto de cores. A intensidade das cores representa a diferença dos dados, cores mais quentes/intensas representam valores mais elevados e pelo contrário, cores mais claras representam valores mais baixos. Este tipo de representação é um bom exemplo de uma metáfora visual orientada a pixel.

- **Metáforas de Projeções Geométricas:** Este tipo de representações são as que nos são mais familiares, estas recorrem a projeções bi e tridimensionais de forma a representar informações relevantes sobre dados multi-dimensionais.

Diagramas de dispersão utilizando coordenadas cartesianas são um bom exemplo deste tipo de metáforas visuais, bem como as coordenadas paralelas.

- **Metáforas Hierárquicas:** Nesta metáfora visual, a exibição dos dados é feita através da divisão do espaço de dados em sub-espacos hierárquicos. Os dados não hierárquicos de  $k$  dimensões são mapeados em sub-espacos bidimensionais, mantendo dessa forma a relação entre eles, aspeto importante na representação dos dados [15]. Esta metáfora visual assume que os dados são hierárquicos, mapeando de seguida a representação gráfica da hierarquia.

Exemplos deste tipo de metáforas visuais são as árvores usadas em organogramas ou em divisões de tarefas, *TreeMaps*.

Tendo em conta as características do *TreeMap*, abordadas na secção 3.8, esta metáfora visual possui potencial para a representação de propriedades inerentes às *toolboxes* e *m-files*. *Toolboxes* têm *m-files* pelo que existe uma hierarquia entre *toolboxes*, alto nível, e *m-files*, baixo nível. Tendo em conta o problema há que estudar como relacionar propriedades como os *concerns* e as *tokens densities* de cada *m-file* com a hierarquia já verificada entre *toolboxes* e *m-files*.

Pretende-se estudar a utilização deste tipo de metáfora visual com a componente de anotações. Como já referido no início do capítulo 3, um modelo SOM é um tipo de anotação sobre os elementos que analisa. Tendo em *Stephanies* deverão poder anotar várias componentes dos SOM. A presença de regiões num modelo SOM

é indicativo da semelhança entre os neurónios constituintes destas regiões. Aos neurónios pertencentes a uma região estão associados os m-files analisados pelo modelo que são mais semelhantes entre si. A relação região, neurónio e m-files apresenta uma estrutura hierárquica também. Dessa forma, a relação presente entre estas estruturas pode ser alinhada a já verificada relação existente entre toolboxes e m-files. Assim, o *TreeMap* deve ser considerado para representação das relações dos dados entre as estruturas indicadas.

### 3.6 Técnicas de Interação

Como já referido, para que se possa tirar o máximo proveito do uso de metáforas visuais, poder permitir a interação do utilizador com as mesmas é necessário de forma a que este possa seleccionar diferentes e determinados aspetos do conjunto dos dados, ou até mesmo da representação visual, de forma a facilitar a detecção dos padrões presentes [14].

De seguida serão apresentadas as técnicas mais relevantes de interação para o problema em estudo.

#### 3.6.1 Filtragem Interativa

Através do uso de filtros um utilizador/analista pode seleccionar subconjuntos de maior interesse para si, do conjunto de dados. A filtragem pode ocorrer diretamente sobre os dados ou através da seleção de propriedades específicas dos dados.

#### 3.6.2 Zoom Interativo

O zoom interativo permite ao utilizador escolher o nível de detalhe com que determinado dado ou conjunto de dados é visualizado. Quando é considerada uma resolução baixa, determinado dado ou conjunto de dados pode ser caracterizado por apenas um pixel, com o aumento da resolução o mesmo dado ou conjunto de dados poderá ser caracterizado por um ícone ou por um outro elemento gráfico. O zoom interativo contempla:

- **Zoom Geométrico:** aumento e diminuição do tamanho dos elementos gráficos, ou seja, dados representados por um pixel passam a ser representados por mais pixels e vice-versa;
- **Zoom Semântico:** contempla a alteração do nível de detalhe da apresentação de um determinado conjunto de dados. Aumentando ou reduzindo o zoom semântico uma nova visualização deve ser apresentada no decorrer da navegação, da herança ou da estrutura de uma árvore, ou da expansão ou redução do volume de informação de dependências em grafos ou matrizes. A navegação de uma visualização exige uma nova visualização, pois a nova possui um significado diferente da anterior, daí o nome zoom semântico [38].

### 3.6.3 Distorção Interativa

Esta técnica permite uma exploração interativa dos dados mantendo ao mesmo tempo a visão geral dos dados, ou seja, apresenta com um nível de detalhe maior determinada área gráfica apresentando ao mesmo tempo as restantes áreas gráficas com menor nível de detalhe [15]. Este tipo de técnicas torna-se muito interessante na medida em que permitem a visualização de uma subárea de mais interesse sem que se perca as relações com o conjunto como um todo.

Pretende-se estudar e posteriormente implementar uma técnica de distorção interativa na elaboração desta dissertação. Na utilização das metáforas visuais, um utilizador deve ser capaz visualizar os códigos *MATLAB* constituintes de um determinado neurónio.

## 3.7 Múltiplas Perspectivas e Múltiplas Visões

Na análise de conjuntos de dados multi-dimensionais são utilizadas múltiplas visões. Cada uma destas, revela uma perspectiva diferente sobre os dados em análise.

Uma visão é uma representação visual de um conjunto de dados. Geralmente, uma única visão não é suficiente por si só para a compreensão efetiva dos dados. Dessa forma, recorre-se a mais que uma visão, o que permite que as conclusões retiradas sobre os dados sejam o mais fundamentadas possíveis. Ao utilizar apenas uma visão como interpretação dos dados pode limitar a interpretação dos dados por parte dos utilizadores, o que pode levar a que estes tomem conclusões precipitadas sobre os dados.

### 3.7.1 Coordenação de Múltiplas Visões

A utilização de múltiplas visões não deve ser feita de uma forma descuidada, ou seja, existem sistemas propostos para o correto uso de múltiplas visões.

Diferentes metáforas visuais devem ser utilizadas para que seja possível obter diferentes perspectivas sobre os dados, mantendo o uso de visões complementares entre si. A análise das diferentes perspectivas dos dados é que permite que padrões e relacionamentos entre os dados sejam descobertos e que de outra forma através de uma análise tradicional não seriam visíveis.

Tendo em conta o contexto do nosso problema a utilização de múltiplas visões é necessário. Os *Joões* devem poder escolher diferentes metáforas visuais de forma a facilitar a compreensão dos resultados visualizados. A liberdade de escolha de quais metáforas um *João* pode visualizar é importante pois dessa forma o mesmo pode adaptar quais as visualizações que mais se adequam à sua necessidade.

### 3.7.2 Visualização e Análise Exploratória dos Dados

Uma visualização gráfica de um conjunto de dados tem como objetivo que o utilizador que está a analisá-la possa compreender de uma melhor forma os dados que estão representados, permitindo consequentemente que a análise dos mesmos se torne mais fácil. No entanto, apenas uma perspectiva sobre dos dados nem sempre e até mesmo poucas vezes, permite uma análise profunda sobre os mesmos, é necessário várias perspectivas para que se possa retirar o maior proveito das relações e dos padrões existentes nos dados.

Mapas coloridos são utilizados como representações visuais dos dados por parte dos SOMs, ou seja, cores são utilizadas como representação dos dados e como ferramenta direta de análise. As representações produzidas pelos SOMs são complexas e pouco convencionais o que torna a compreensão dos dados e das relações presentes entre estes um processo complicado. Aliás, uma das principais motivações do projeto que enquadra esta dissertação é estudar até que ponto *João* conseguem interpretar estas visualizações comparativamente com alternativas mais clássicas e de que forma pode-se auxiliar a compreensão das mesmas. Assim, existe a necessidade de perceber quais as melhores estratégias a utilizar na análise deste tipo de representações.

Existem duas representações dos dados produzidas pelo SOM, sendo elas a *U-Matrix* e as *Component Planes*. Cada uma destas visualizações é representada como um mapa colorido, sendo por isso categorizadas como metáforas visuais do tipo orientadas a pixels.

De forma a compreender como devem ser interpretadas a *U-Matrix* e as *Component Planes* dentro do contexto do nosso problema, três figuras são consideradas, Figura 3.2 para a *U-Matrix* e as Figuras 3.3 e 3.4.

- ***Unified Distance Matrix***: Denominada abreviadamente como *U-Matrix*, esta visa a representar a distância entre *neurónios*.

A cada unidade, neurónio, a distância média *Euclidiana* é calculada entre o seu vetor de unidade SOM com os vetores das 8 unidades SOM que são seus vizinhos diretos. Dependendo do valor dessa média, aos neurónios são lhes atribuídos uma cor tendo em conta a proximidade entre os mesmos e entre cada neurónio vizinho.

As cores vermelhas representam neurónios com distâncias *Euclidianas* máximas em relação à sua vizinhança. São neurónios com grandes variabilidades de valores. As regiões compostas por este tipo de cores, identificadas na Figura 3.2 por Z, devem ser interpretadas como separações de *clusters*. Pelo contrário, regiões representadas com cor azul, na Figura 3.2 identificadas por A1, A2, A3 e C, correspondem a regiões homogêneas. Estas possuem distâncias médias *Euclidianas* mínimas entre si, sendo por isso representativas de *clusters*. São conjuntos de neurónios que possuem valores muito semelhantes entre si. Por último, regiões verdes representam valores

intermédios de distância entre as unidades SOM. Falta agora considerar regiões azuis com um verde subtil associado, identificadas na Figura 3.2 por *B*, *E1*, *E2* e *D*. Tais regiões são consideradas como relativamente homogéneas. No entanto, os dados que codificam estas regiões traem vários padrões de *concerns* bem definidos.

A utilização desta visualização é fundamental para perceber a estrutura subjacente dos dados quando o utilizador não tem conhecimento prévio sobre os agrupamentos dos dados, fornecendo uma representação global dos dados.

De forma a analisar a informação relativa aos vários *concerns* consideradas é necessário estudar o segundo tipo de representação que os SOM oferecem, as *Component Planes* [26].

- ***Component Planes***: Este tipo de representação gráfica dos dados permite uma visualização da densidade dos dados de entrada. Tendo em conta o nosso problema, as *Component Planes* representam um determinado *concern*. Cada *Component Plane* corresponde à análise, no contexto do nosso problema, de um *concern* específico. Este tipo de visualização visa a representar a distância entre valores do mesmo *concern*. Regiões com cor azul representam uma presença baixa do *concern* e valores mais altos são representados pela cor vermelha.

De forma a aproveitar ao máximo o uso da metáfora visual *Component Planes* é necessário analisá-la em simultâneo com as restantes visualizações *Component Planes* dos outros *concerns* e com a *U-Matrix*. Através da comparação entre *Component Planes* é possível inferir a existência de relações entre diferentes *concerns*. Se as representações de duas ou mais *Component Planes* forem similares (ambas possuem valores altos ou baixos na mesma região) quer dizer que existe uma relação de semelhança entre as mesmas, caso o oposto se verifique, significa que estas estão relacionadas de uma forma inversa (valores baixos de um determinado *concern* corresponde a valores altos de outro *concern* e vice-versa). No entanto pode-se verificar também que nenhuma relação aparente existe entre os dados.

A análise singular de cada uma das visualizações, como referido, permite a obtenção de conhecimento adicional. No entanto, apenas a análise composta de ambos os tipos visualizações permite o aproveitamento máximo destas duas metáforas visuais.

É observável nas representações *Component Planes* Figura 3.3 e Figura 3.4 a partilha de duas regiões comuns vermelhas, em baixo, do lado esquerdo e do lado direito, o que permite saber que existe uma correlação significativa entre os *concerns* representados nas



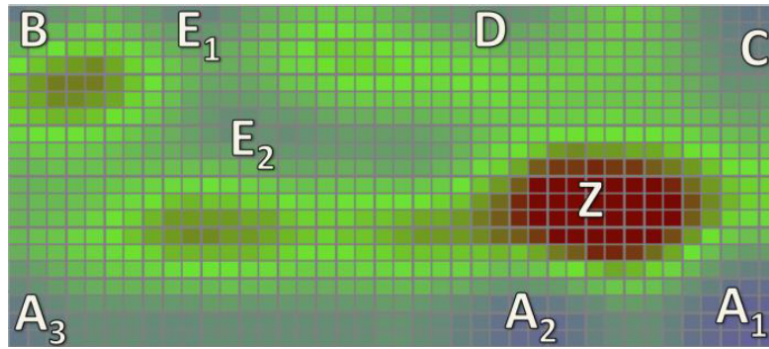
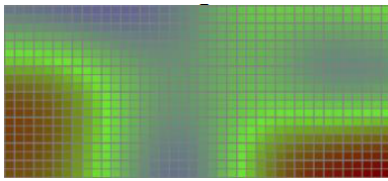
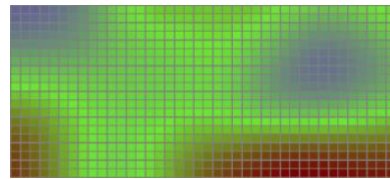


Figura 3.2: Visualização U-Matrix [26]

Figura 3.3: Component Plane referente à *concern* Verificação de argumentos de funções e valores retornados [26]Figura 3.4: Component Plane referente à *concern* Verificação do tipo de dados [26]

*Component Planes*. Para além deste facto, estas duas regiões nas *Component Planes* correspondem a regiões azuis na representação *U-Matrix* o que significa que esta correlação entre as duas *Component Planes* é uniforme ao longo dos dados de entrada.

A partilha de regiões vermelhas nas *Component Planes* numa região vermelha da *U-Matrix* em vez de regiões azuis representa distâncias dispares entre os *neurónios* dessas regiões. A sobreposição das representações nestes casos, provavelmente não seriam representativas dos dados como um todo, pelo que as conclusões inferidas sobre a análise desses nestas situações seriam contribuições negativas na análise dos mesmos.

A inexistência de uma região azul na *U-Matrix* sem correspondência de uma região vermelha numa das *Component Planes* é bom sinal pois a ocorrência de tal situação representaria um conceito difuso, provavelmente irrelevante na análise.

### 3.8 TreeMap

O *TreeMap* é o termo inglês para Mapa em Árvore, o qual é uma metáfora visual do tipo hierárquica, como referido na secção 3.5.1.

A metáfora *TreeMap* é particularmente adequada na visualização de grandes estruturas hierárquicas, mapeando diretamente uma hierarquia através da subdivisão recursiva de retângulos juntos uns aos outros, como é possível observar na Figura 3.5.



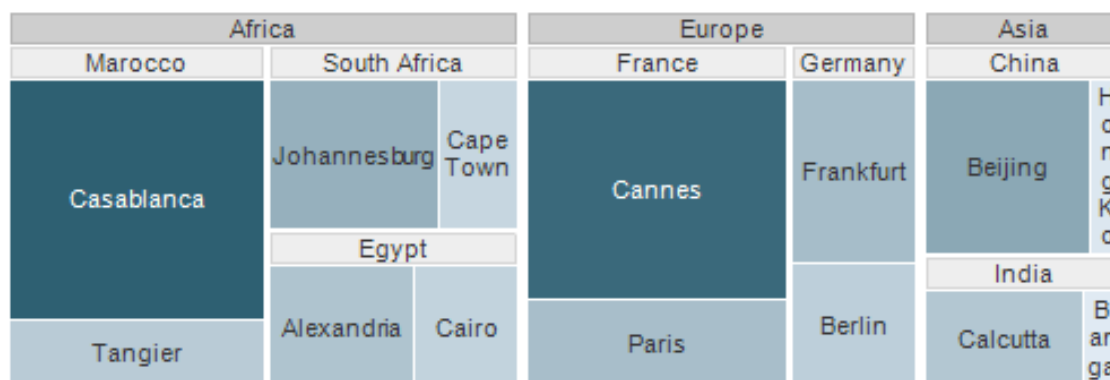


Figura 3.5: Exemplo ilustrativo de um Treemap [7]

Um *Treemap* diz-se que possui ramos, sub-ramos e nós folha. Cada ramo da árvore na representação consiste num retângulo, o qual é composto por um conjunto de pequenos retângulos, sendo estes denominados de sub-ramos. Um retângulo nó folha tem uma área proporcional a uma dimensão específica dos dados, ou seja, através da área ocupada na representação, a percentagem da presença desta dimensão nos conjuntos de dados é indicada [43] [6]. Normalmente os nós folha são coloridos de uma cor diferente de forma a representarem partes distintas dos dados. Quando a cor e o tamanho da dimensão estão correlacionados de alguma forma com a estrutura da árvore, um analista na maior parte das vezes consegue facilmente ver padrões que de outras formas seriam difíceis visualizar. Uma segunda vantagem dos *Treemap* é, por construção, eles são eficientes no uso do espaço. Dessa forma, é possível representar de forma legível milhares de itens no ecrã simultaneamente [43]. O que faz com que *Treemaps* sejam uma opção eficiente na representação de hierarquias, oferecendo uma rápida visão geral da estrutura dos dados [6].

Na Figura 3.5 ilustra um exemplo de um *Treemap*, a qual a sua interpretação facilita perceber o seu funcionamento. Neste exemplo, cada retângulo representa uma cidade e são coloridos e atribuídos aos retângulos um tamanho segundo um parâmetro *Vendas*. Num *Treemap* o tamanho dos retângulos diminui do topo superior esquerdo para o inferior direito de um mapa. Assim, é observável que neste caso, o continente África possui o maior número de vendas e a Ásia o menor [8].

Após a contextualização ao *Treemap* podemos agora considerar o nosso caso de estudo nesta metáfora.

Tendo em conta as características de um *Treemap*, como já referido na secção 3.5.1, o estudo das anotações enquadra-se nesta metáfora. Dessa forma, tem de se estudar como deve ser estruturado o *Treemap* tendo em conta as anotações. Uma hipótese a estudar é considerar como parâmetro de análise nesta metáfora as anotações presentes num conjunto de dados. Retângulos devem possuir uma hierarquia de modelos, regiões e blocos de código com as anotações respetivas associadas.

### 3.9 Como escolher que Visões utilizar?

Segundo [Maletic e colegas (2002)] é necessário considerar cinco dimensões para se poder conceber soluções de visualização de *software*:

- **Atividades:** Fim a que a visualização se destina.
- **Audiência:** A quem se destina a visualização.
- **Alvo:** Qual a fonte de dados a ser representada.
- **Representação:** De que forma o conjunto de alvos será representado.
- **Meio:** Mecanismo através do qual a visualização será apresentada.

As cinco dimensões podem ser adequadas ao nosso trabalho. Dessa forma, relativamente a este trabalho, pode-se já salientar:

- **Atividades:** Possibilitar a pesquisa tendo em conta o modelo de *datamining* utilizado (repositório e atual relação com as regiões e neurónios sobre um modelo SOM previamente anotado).
- **Audiência:** Programadores de código *MATLAB* que não necessitam de ser necessariamente conhecedores de mapas-autorganizados (os utilizadores *Joes*).
- **Alvo:** Programas *MATLAB* que têm problemas de *concerns* no repositório estudado [26].
- **Representação:** Visualização de código e *TreeMaps*.
- **Meio:** Proposta e desenvolvimento de componentes para o protótipo *web*.

## FRAMEWORKS E TECNOLOGIAS PARA DESENVOLVIMENTO WEB

Tendo em conta os diferentes requisitos que uma aplicação pode ter, o mundo das *frameworks* oferece soluções para os mesmos, dessa forma à que fazer uma análise crítica às variadas *frameworks* que existem no mercado de forma a escolher as que mais se adequam a responder às necessidades do problema em questão. Este capítulo divide-se no estudo das tecnologias necessárias para desenvolver o *backend* secção 4.1 e o *frontend* secção 4.2 de uma aplicação *web*. No âmbito do *backend* o tema sistema de base de dados, sub-secção 4.1.1, é abordado bem como uma *DML* (*Data Manipulation Language*), sub-secção 4.1.2, para manipulação dos dados da base de dados e *frameworks*, sub-secção 4.1.4, para implementação do *backend*. Em relação ao *frontend* são abordadas bibliotecas e tecnologias para a construção gráfica da aplicação. Para construção gráfica da página *web* o CSS e a biblioteca Bootstrap são abordados, sub-secção 4.2.1, e bibliotecas *JavaScript* são apresentadas para a representação dos resultados, sub-secções 4.2.2, 4.2.3 e 4.2.4.

### 4.1 Backend

No desenvolvimento de uma aplicação *web* uma base de dados é um componente fundamental. Dados do cliente e os dados de suporte à aplicação *web* são armazenados numa base de dados. No desenvolvimento de uma aplicação informática a escolha das ferramentas auxiliares a usar é uma decisão importante. Os requisitos inerentes à aplicação influenciam as *frameworks* que devem ser utilizadas. O *Backend* trata da recolha de dados e do tratamento dos mesmos, enviando os de seguida para uma *view MVC* do sistema.

#### 4.1.1 Base de Dados

Uma base de dados é uma coleção de dados que está organizada de forma a que possa ser acedida facilmente, “gerenciada” e atualizada [33]. Existem dois grandes tipos de sistemas de bases de dados, os sistemas de base de dados relacionais e os não relacionais.

O modelo relacional de base de dados é baseado no conceito matemático de relação e é muitas vezes implementado como uma tabela. Uma base de dados relacional baseada neste modelo é definida utilizando regras baseadas em matemática que indicam precisamente que tabelas devem ser incluídas. Tuplos são compostos por dados em linhas de uma tabela. Os dados são armazenados em várias tabelas em que cada uma corresponde a um conjunto de tuplos para atributos predefinidos.

Uma base de dados não relacional é uma base de dados que não pode ser relacionada por expressões matemáticas. Exemplos de aplicações de bases de dados não relacionais são sistemas de bases de dados de, projetos e produção auxiliado por computadores, engenharia de software, sistemas de informação multimédia, dados gráficos, dados de som, dados documentos de texto e ainda sistemas especialistas de bases de dados.

Por exemplo, uma pesquisa por correspondência de padrões para dados gráficos é muito complicada. Um típico sistema de base de dados relacional está longe de ser capaz de realizar uma análise gráfica por correspondência de padrões. Tal pesquisa pode exigir a modificação do interpretador de linguagem de consulta da base de dados relacional.

Bases de dados que contêm documentos de texto em inglês podem exigir uma pesquisa mais complicada do que uma simples comparação de texto dos dados. Além disso, um mecanismo de pesquisa alternativo seria necessário para pesquisar texto em idioma diferente do inglês, como documentos em japonês. Pesquisas através de bases de dados de texto podem exigir:

- comparação de documentos ou formatos que não sejam texto simples;
- a capacidade para lidar com palavras soletradas de forma diferente ou que tenham entrado de forma incorreta na base de dados;
- a capacidade de procurar palavras que podem ser incluídas no documento de texto em vez de palavras na consulta;
- Diferentes tempos verbais das mesmas palavras.

Texto de linguagens que não são facilmente representadas em texto simples muitas vezes torna-se difícil implementar num sistema de base de dados relacionais.

### 4.1.2 MySQL como DML

Uma linguagem de manipulação de dados, ou linguagem de consulta, para um sistema de base de dados, tipicamente especifica operações sobre uma ou mais relações para criar uma nova relação. I.e, uma operação de restrição forma um subconjunto de linhas numa tabela através de aplicar uma condição ou um predicado, aos valores em cada linha. Uma operação de projecção remove colunas das linhas através de formar um fluxo de linhas que contém determinadas colunas. Uma operação junção combina dados de uma tabela com dados de uma outra tabela, tipicamente através da aplicação de um predicado comparativo de valores numa coluna da segunda tabela.

O software *MySQL* oferece um servidor de base de dados muito rápido, *multithreaded*, multi utilizadores e com um *SQL* robusto [28]. Um servidor *MySQL* destina-se para missões-criticas, produção elevada de sistemas, bem como para incorporação em software implementado em massa. A *Oracle* é uma marca registada da *Oracle Corporation* e/ou dos seus afiliados. A *MySQL* é uma marca da *Oracle Corporation*.

O *MySQL* suporta muitos tipos de dados: inteiros assinados/não assinados 1, 2, 3, 4, e 8 *bytes* de comprimento, *FLOAT*, *DOUBLE*, *CHAR*, *VARCHAR*, *BINARY*, *VARBINARY*, *TEXT*, *BLOB*, *DATE*, *TIME*, *DATETIME*, *TIMESTAMP*, *YEAR*, *SET*, *ENUM*, e tipos espaciais *OpenGIS* [29].

Em termos de segurança tem implementado um sistema de privilégios e de *password* que é flexível e seguro, com verificação baseada em *host* ativa. Tem segurança da *password* por encriptação de todo o tráfico de *passwords* nas conexões a um servidor.

O *MySQL* suporta grandes bases de dados. Em termos de escalabilidade e em termos de limites de performance, a *MySQL* utiliza o servidor *MySQL* com bases de dados que contém 50 milhões de registos. A *MySQL* sabe que existem utilizadores que utilizam o servidor *MySQL* com 200,000 tabelas e cerca de 5,000,000,000 linhas.

*APIs* para *C*, *C++*, *Eiffel*, *Java*, *Perl*, *PHP*, *Python*, *Ruby*, e *Tcl* estão disponíveis, o que possibilita que clientes *MySQL* possam ser escritos em várias linguagens.

Suporte completo para vários conjuntos diferentes de caracteres, incluindo o *latin1* (*cp1252*), *german*, *big5*, *ujis*, vários conjuntos de caracteres *Unicode*, e outros mais. I.e., os caracteres Escandinavicos “å”, “ä” e “ö” são permitidos em tabelas e em nomes de colunas.

O *MySQL* inclui vários programas para o cliente e programas utilitários. Estes incluem tanto programas da linha de comandos tais como o *mysqldump* and *mysqladmin*, e programas gráficos tais como o *MySQL Workbench*.

O *phpMyAdmin* é uma ferramenta de software livre escrita em *PHP*, destinada a lidar com a administração do *MySQL* através da *web*. O *phpMyAdmin* suporta uma ampla gama de operações no *MySQL*. Operações frequentemente utilizadas (gerenciamento de bases de dados, tabelas, colunas, relações, índices, usuários, permissões, etc) podem ser realizadas através da interface do usuário. Mais, qualquer uma destas ações pode também ser realizada em *SQL* diretamente no *phpMyAdmin*.

Uma das principais razões pelo qual escolhemos utilizar uma base de dados *MySQL* foi a quantidade de registos a guardar na base de dados. O repositório atual contém 35208 m-files, quando decomposto nos modelos de dados considerados, o número de registos é na ordem dos milhões. Uma base de dados *MySQL* consegue conter 50 milhões de *records*, pelo que para o repositório considerado e para futuros é uma boa solução tendo em conta que é uma solução *opensource*.

### 4.1.3 SQL e Queries

*SQL* é uma linguagem tradicional para armazenar, manipular e ir buscar dados a bases de dados [41].

A *query* é uma operação de consulta na base de dados. Uma *query* tem uma sintaxe diferente consoante o tipo de manipulação de dados a que esta se destina [10].

- Recolher dados para consulta:

Listagem 4.1: Consulta de dados em *SQL*

```
1  SELECT coluna1
2  FROM nome_tabela
3  WHERE coluna2 = valor1;
```

Através da conjugação de diferentes tipos de operações sobre a base de dados *queries* poderosas podem ser construídas de forma a visualizar dados que sem o relacionamento dos dados numa base de dados seria difícil observar.

### 4.1.4 LARAVEL

*LARAVEL* aparece no mercado como uma das *framework's* em *PHP* mais populares da atualidade [20]. É uma ferramenta *open source*. Oferece ferramentas necessárias ao desenvolvimento de aplicações robustas e com uma elevada dimensão [30].

O principal objetivo do *LARAVEL* é permitir que o programador que a utilize possa trabalhar de uma forma estruturada e rápida [19].

O *LARAVEL* utiliza o *Composer* para administrar as dependências, este permite gerir de uma forma fácil os pacotes terceiros da aplicação *web* [19]. Ferramenta intuitiva de utilizar e com uma vasta documentação de suporte e fóruns dedicados ao mesmo.

Permite uma implementação rápida de um sistema de logins completo. Possui um compilador de templates *Blade*, que visam facilitar o desenvolvimento do *frontend*. Este permite o uso de *PHP* puro com a sintaxe do *template*. O principal objetivo do *Blade* é reduzir a quantidade de código *PHP* presente no *HTML* e aumentar o reuso de código.

Possui um *QueryBuilder*, que permite uma notação simples na construção de *queries*, através do uso deste não é necessária a escrita das mesmas em *SQL*, o *QueryBuilder* fornecido tem uma sintaxe própria, simples e intuitiva.

O *LARAVEL* oferece também a possibilidade de criar ficheiros de migração referentes à estrutura da base de dados da aplicação. Através de comandos *LARAVEL* é possível fazer a criação de tabelas e afetar *constraints* às mesmas num ficheiro, podendo este posteriormente ser importado na base de dados recorrendo a mais um comando.

Esta *framework* possui um estilo *MVC* arquitetural por base. O *MVC* é usado de forma a separar os conceitos *Model* (Modelo de Dados), *View* ( Vista ) visualização dos dados e *Controller* (Controlador de Dados) que atua tanto no *Model* como na *View*, controlando o fluxo de dados para o modelo de dados e atualizando a *view* cada vez que o modelo de dados é atualizado.

Existem ficheiros do *LARAVEL* que são ficheiros representativos das tabelas presentes na base de dados. O programador pode programar funções para que mais tarde possam ser chamadas nos Controladores. Estas funções permitem simplificar a representação das relações entre tabelas que possuem chaves estrangeiras entre si.

O *PHP* é a linguagem e *LARAVEL* a *framework* com a qual se possuía mais experiência.

## 4.2 FrontEnd

O *Frontend* diz respeito à interface gráfica com que um utilizador numa página *web* interage. Como linguagem de estrutura de documento (DOM) um *frontend* utiliza normalmente o *HTML*. Para realizar a estilização de elementos *HTML* o *CSS* é utilizado. Para a página *web* ser interativa é necessário *JavaScript* para tratar dos eventos a criar na página.

### 4.2.1 CSS e Bootstrap

CSS é uma linguagem de estilo para estilizar uma página *web*. Já o *Bootstrap* é um kit de ferramentas *open source* para desenvolvimento com *HTML*, *CSS*, e *JS* (JavaScript). [2]. Ao longo da sub-secção, características, funcionalidades e módulos do *Bootstrap* são apresentados.

De forma a manter a responsividade da página e alocar determinado espaço da página aos componentes o *Bootstrap* oferece um conjunto de propriedades interessantes. O *Bootstrap* o tamanho das páginas *web* como se fossem constituídas por um conjunto de 12 colunas.

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
# of columns	12				
Gutter width	30px (15px on each side of a column)				
Nestable	Yes				
Column ordering	Yes				

Figura 4.1: Propriedades do sistema de colunas do Bootstrap

Utilizando os vários tipos de classes *col* é possível ter diferentes perspectivas da mesma página *web* consoante a resolução do dispositivo em que este está a ser utilizado, permitindo ajustar o *layout* da página a cada tipo de resolução diferente. Um componente pode ocupar um número diferente de colunas consoante a resolução em que a página *web* está a ser visualizada através da utilização de classes CSS específicas.

Botões *Bootstrap* vêm com classes personalizadas para terem um formato apelativo e para que a cor do botão, tamanho de letra ou do mesmo, e o estado, se ativo ou não, possa ser escolhida através do uso de uma classe. Os botões são utilizados em ações de formulários, diálogos e outras coisas.

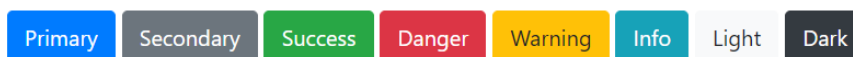


Figura 4.2: Botões do Bootstrap

O *plug-in Popover* é semelhante a um kit de ferramentas, é uma caixa *pop-up* que aparece quando o utilizador clica (evento normal associado ao despoletar de um *popover*) num elemento. A diferença entre um *popover* um *pop-up*, é que o primeiro pode conter muito mais conteúdo. *Popovers* utilizam a biblioteca terceira *Popper.js* para o seu posicionamento na página.



O *plug-in Modal* é uma caixa de diálogo/janela *popup* exibida. Esta janela é um *pop-up* do *JavaScript* leve e multifuncional que é personalizável e responsivo. Este é exibido através de um evento. É necessário um botão ou *link* para que o *modal* seja exibido. Os *modais* são criados com *HTML*, *CSS* e *JavaScript*. Quando exibidos posicionam-se sobre todo o restante documento.

As *navs* são componentes normalmente utilizados numa barra *header* de uma aplicação *web*. No entanto conjugando um pouco de *JavaScript* é possível criar painéis dinâmicos a representação visível é alterada consoante um *click* numa aba diferente.

É visível pelo par de Figuras 4.5 e 4.6 que clicando numa aba conteúdo diferente no mesmo *container*.

### 4.2.2 Javascript, D3 e ZingChart

Como em qualquer outra aplicação *web*, a linguagem *JavaScript* será utilizada para tratar os eventos despoletados pelo o utilizador.

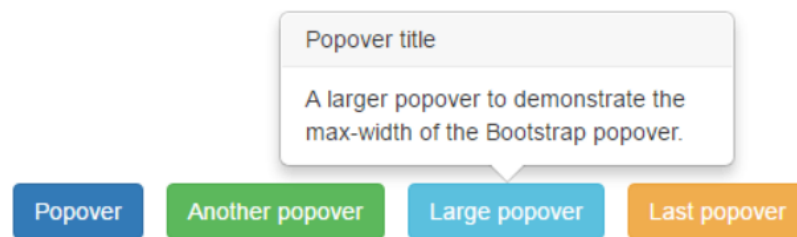


Figura 4.3: Exemplo visual de um Popover

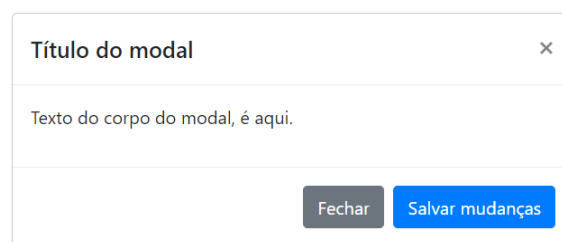


Figura 4.4: Exemplo visual de um Modal

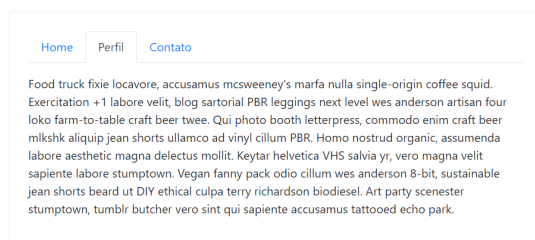


Figura 4.5: Caso interessante da utilização de um componente nav versão a

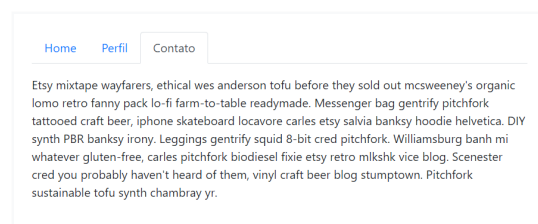


Figura 4.6: Caso interessante da utilização de um componente nav versão b

Para a implementação das metáforas visuais na aplicação *web* é necessário utilizar uma ferramenta dedicada.

O *D3* é uma biblioteca *JavaScript* para produzir visualizações de dados interativas em páginas *web*. Utiliza os padrões amplamente implementados, *Scalable Vector Graphic* (*SVG*), *HTML5* e *CSS* [3]. Esta biblioteca, permite um grande controlo sobre o resultado visual final. Este, é normalmente utilizado na criação de gráficos interativos para novas aplicações *web*, *dashboards* para visualização de dados e produção de mapas [44].

A biblioteca *D3* utiliza funções *JavaScript* pré-definidas para selecionar elementos, criar objetos *SVG*, estilizá-los, adicionar transições e efeitos dinâmicos aos mesmos. Grandes conjuntos de dados podem ser associados a objetos *SVG* usando funções simples do *D3.js* de forma a gerar gráficos e diagramas altamente customizáveis [31]. Os dados de entrada para estas representações gráficas podem estar em vários formatos, sendo eles geralmente *JSON*, *CSV* ou *geoJSON*. No entanto, se necessário, funções *JavaScript* podem ser desenvolvidas de forma a ler os dados em formatos diferentes dos indicados.

A biblioteca *ZingChart* à semelhança da biblioteca *D3* utiliza funções *JavaScript* para visualização de dados em contexto *web*. A biblioteca *ZingChart* oferece uma vasta gama de representações gráficas de dados. Cada gráfico possível de implementar é customizável num grande leque de propriedades, tais como, customização dos eixos, tratar eventos associados à interação com o gráfico e muitas mais.

Para cada tipo de gráfico é necessário fornecer uma estrutura dos dados específica para que esta seja interpretada e posteriormente representada pela biblioteca *ZingChart*.

Na Listagem 4.2 está representada a estrutura hierárquica necessária à construção de *TreeMaps* segundo a biblioteca *ZingChart*. É uma estrutura que por cada nível dá origem a um novo conjunto de elementos filhos do nível pai.

Listagem 4.2: Exemplo *ZingChart* da estrutura de dados para um *TreeMap*

```
1 {
2   "type": "treemap",
3   "series": [{ "text": "North_America",
4                 "children": [{ "text": "United_States",
5                               "children": [{ "text": "Texas",
6                                              "value": 21
7                                              }, {
8        ...
9        ]
10      }]}
11   ]
12 }
```

	Detecção automática de linguagens	Highlight das expressões relevantes na linguagem	Definição de novas linguagens	Highlight de <i>Tokens</i>	Suportar Interatividade
highlight.js	SIM	SIM	SIM	SIM	SIM
PRISM	SIM	SIM	SIM	SIM	SIM

Tabela 4.1: Lista de características necessárias das ferramentas

Na Listagem 4.3 está representada a estrutura necessária à montagem de *HeatMaps* segundo a biblioteca *ZingChart*. Cada inteiro pertencente a um *array* corresponde ao valor de uma célula constituinte do *HeatMap*. Esse valor determina a cor representativa da célula em questão.

Listagem 4.3: Exemplo ZingChart da estrutura de dados para um *HeatMap*

```

1 {
2   "type": "heatmap",
3   "series": [
4     { "values": [59, 15, 5, 30, 60, 99, 28] },
5     { "values": [34, 32, 87, 65, 9, 17, 40] },
6     { "values": [90, 19, 50, 39, 12, 49, 14] }
7   ]
8 }
```

### 4.2.3 Highlight e PRISM

A *highlight.js* e a *PRISM* são ambas uma biblioteca *JavaScript* que realçam palavras. As palavras são realçadas se estas forem relevantes no contexto semântico da linguagem, ou seja, palavras e expressões reservadas, numa determinada linguagem. Ambas as ferramentas estão preparadas para fazer o *highlight* num conjunto de linguagens de programação, das quais o MATLAB faz parte. As bibliotecas têm um analisador de sintaxe e semântica que permite identificar quais as palavras/expressões relevantes em diferentes contextos de programação. Ou seja, o papel de ambas as bibliotecas é o papel de qualquer editor de código que um programador utiliza no seu computador.

No entanto, como em qualquer ferramenta, estas bibliotecas têm pontos fortes e fracos. Assim é necessário analisar quais as funcionalidades de interesse que as bibliotecas têm a oferecer e se as mesmas satisfazem as necessidades para as quais são necessárias. Verificar de igual forma também, se as necessidades que as ferramentas não satisfazem podem ser colmatadas por outras ferramentas ou através de um outro processo.

Após análise de ambas as bibliotecas em contexto real na dissertação retirou-se as informações apresentadas na Tabela 4.1. Ambas as ferramentas correspondem aos requisitos necessários para desempenhar as funções pelas quais foram escolhidas. No entanto, só faz sentido o uso de uma destas ferramentas. Por isso mesmo, a ferramenta utilizada no protótipo é a *highlight* pois a sua implementação e implementação de funcionalidades sobre a ferramenta revelaram-se acessíveis.

#### 4.2.4 Summercode e Quill

*Summercode* e *Quill* são ambas bibliotecas de *JavaScript* e *CSS* que permitem a implementação de um editor de texto rico num ambiente *web*. Ambos enquanto editores ricos de texto possuem as funcionalidades base:

- De formatação e edição de texto;
- Inserção de conteúdo multimédia e de hiperligações;

E por último, a característica mais interessante das ferramentas tendo em conta o contexto do problema:

- suporta linguagem *HTML* e *CSS*.

Esta característica é o porquê do uso deste tipo de ferramentas. Tendo em conta as anotações e de forma a enriquecê-las, um sistema de referências a elementos do m-file foi criado.

## ARQUITETURA PROPOSTA

Este capítulo tem como objetivo apresentar uma arquitetura de uma ferramenta de análise exploratória de bases de código *MATLAB*. No capítulo são apresentadas as características necessárias a cada componente da arquitetura. Na secção 5.1 é descrita a arquitetura global do sistema. Nas secções 5.2, 5.3, 5.4, 5.5, a arquitetura de cada componente é detalhada, o respetivo modelo de dados que representa cada componente é abordado bem como as principais funcionalidades e restrições de cada um. A última secção do capítulo apresenta o modelo de dados implementado sobre uma base de dados *MySQL* utilizado no protótipo.

### 5.1 Arquitetura de uma ferramenta de estudo de uma linguagem de programação

O protótipo foi desenvolvido com o propósito de acomodar num só lugar todos os tipos de dados inerentes ao processo de análise do código *MATLAB*.

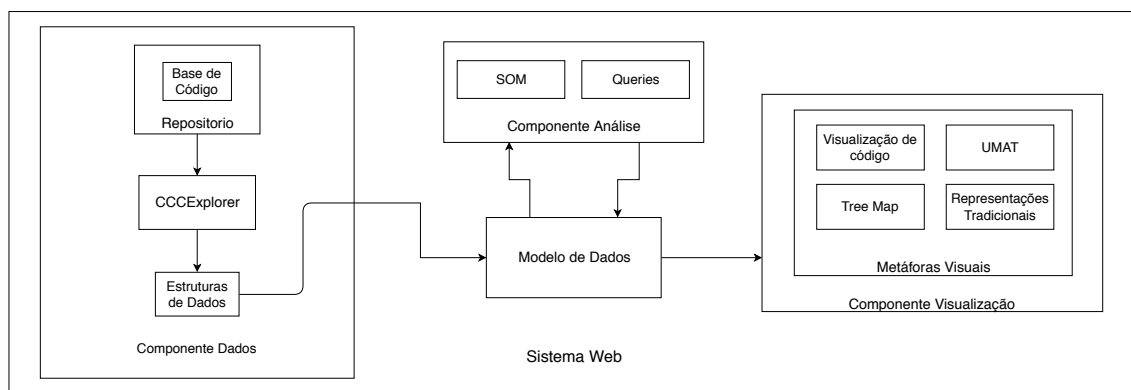


Figura 5.1: Arquitetura Global do Sistema

A Figura 5.1 representa a arquitetura global da solução proposta. A arquitetura é constituída por três componentes principais relacionados por um modelo de dados. A arquitetura abordada neste trabalho engloba as etapas características de uma ferramenta de análise: tratamento de dados, análise dos dados e representação das análises efetuadas.

A componente dados produz estruturas de dados que alimentam o modelo de dados, através da ferramenta *CCCExplorer* (ver secção 2.5), que extrai dados do repositório, incluindo métricas (ver secção 6.2.2). A componente análise alimenta-se do modelo de dados e alimenta por sua vez o modelo de dados com os dados resultantes. Por sua vez, o componente visualização apresenta os dados sob a forma de metáforas visuais (ver secção 3.5).

## 5.2 Componente de identificação e recolha de dados sobre um repositório de código MATLAB

Este componente efetua o tratamento dos dados. O código *MATLAB* proveniente de um repositório é decomposto em estruturas de dados de análise para o sistema de *queries* e dele extraídas métricas para subsequente análise (dele, o código *MATLAB*, bem entendido).

Nesta solução, o *CCCExplorer* é utilizado como analisador léxico que alimenta o modelo de dados com as sequências de *tokens* provenientes da base de código *MATLAB* do repositório em estudo (Figura 5.1). O *CCCExplorer* foi já utilizado em trabalhos anteriores [26] enquanto ferramenta utilizada na extração de métricas para o SOM. Esta ferramenta foi melhorada e adequada ao trabalho necessário, passando a contemplar funcionalidades úteis para alimentar a base de dados relacional. Na sua nova versão o componente *CCCExplorer* executa a seguinte sequência de tarefas:

1. Identificação dos *tokens* (ver subsecção 2.4.3) presentes no repositório;
2. Decomposição do código *MATLAB* em estruturas de dados representando sequências de *tokens* e em informações complementares para a análise do código;
3. Produção do *output* do processo anterior em formato *SQL* (ver subsecção 4.1.3) para integração numa base de dados.

A figura 5.2 representa de uma forma esquemática as etapas constituintes do componente e que devem ser realizadas pelo *CCCExplorer*.

A primeira etapa é a leitura do código *MATLAB*. O código está agrupado em *m-files*, que são ficheiros de texto. Dessa forma, o conteúdo destes é lido através de um objeto *fileReader*. Para cada repositório de estudo o *CCCExplorer* identifica todos os *tokens* distintos presentes. Neste trabalho todos os *tokens* presentes no repositório são considerados

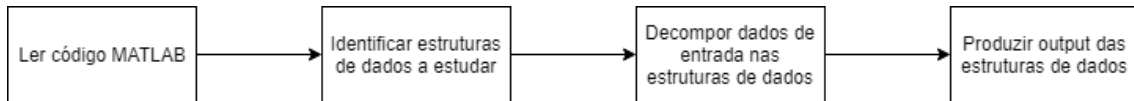


Figura 5.2: Esquema representativo dos processos a completar pelo CCCExplorer

no estudo do código *MATLAB*, ver secção 2.4.2. Contrariamente ao que foi considerado em trabalhos anteriores [17] [26] [32] [26], onde apenas os *tokens* da Tabela 2.2 eram considerados como dados de análise, todos os *tokens* presentes no código *MATLAB* são considerados como dados de estudo, especialmente como dados de análise para o módulo de *queries* 6.6.

Uma instância *token* refere-se à ocorrência individual de um determinado *token*. Por exemplo, as várias ocorrências no código da palavra reservada “*while*” são interpretadas como instâncias do *token* “*while*”. Para análises realizadas pelo sistema de *queries* a posição das instâncias *tokens* é um dado necessário, este tópico é abordado na sub-secção 5.5.1 e no tópico 6.6.1.

Após a identificação do conjunto de *tokens* presentes no repositório todas as instâncias *tokens* são transformadas em dados de estudo de forma a manter a informação específica sobre cada m-file. Esta etapa é denominada como decomposição na medida em que o conteúdo *MATLAB* de um m-file é decomposto num conjunto de estruturas de dados que são abordadas em maior detalhe na secção 5.5.

A última etapa do processo é a produção de um *output* da decomposição realizada. Visto que este *output* é integrado numa base de dados o formato do *output* é *SQL*. O *output* produzido alimenta o modelo de dados, concluindo assim os processos do componente dados, ver Figura 5.1.

### 5.3 Componente de análise sobre o repositório MATLAB

O sistema utiliza ferramentas de análise que processam os dados produzidos na componente anterior, sendo ela o modelo de *queries*. O componente abordado nesta secção deve:

1. Utilizar o modelo SOM estudado [26] como ferramenta de análise do repositório;
2. Utilizar *queries* como ferramenta de análise sobre o repositório.

O SOM (ver secção 3.1) utiliza a métrica *Token Density* (secção 3.1.1) ( número de *tokens* de um *concern* por linhas de código de um m-file [26]), como *input* de dados de treino. O estudo do modelo SOM referido [26] serve dessa forma como uma análise sobre o repositório total armazenado na base de dados.

Para treino de modelos SOM, o sistema contempla o *UbiSOM* (ver secção 3.1) como ferramenta de análise por permitir o *streaming* dos dados de análise no treino de mapas. O *design* do sistema permite uma entrada contínua e posterior armazenamento de m-files. Dessa forma, é possível gerar novos mapas sobre modelos antigos que contemplam a introdução de novos dados de análise. Mais uma vez, embora não se modelem novos modelos SOM no protótipo, foi contemplado no modelo de dados este conceito.

As *queries* são uma ferramenta de validação dos SOM e uma ferramenta de análise do repositório sobre instâncias *word tokens* (neste caso, conjunto dos *word tokens* pertencentes à Tabela 2.2) e/ou expressões (conjunto de instâncias tokens). Estas *queries* estudam as relações entre *tokens* e a relação *Token - Concern* (ver subsecção 2.4.3).

Na secção 5.5, secção modelo de dados, é abordado de forma mais detalhada como o modelo SOM estudado [26] é representado no modelo relacional e como os módulos de *queries* estão estruturados.

## 5.4 Componente de representação dos dados analisados

Este componente é responsável pela a visualização de dados. Um conjunto de módulos (metáforas visuais)?? representam os resultados das análises realizadas sobre o repositório *MATLAB*. O componente de visualização deve:

1. Ter visualização de código;
2. Utilizar a metáfora visual *TreeMap* (secção 3.8) para análise global do repositório e como análise sobre modelos SOM.
3. Utilizar estratégias visuais de auxílio à compreensão dos dados.
4. Utilizar a biblioteca *Bootstrap* (subsecção 4.2.1) para estilizar os componentes de forma a que estes sejam intuitivos na sua utilização.

No protótipo, diferentes dados têm de ser representados. A visualização de código é a metáfora principal a utilizar no sistema, tendo em conta que os dados a analisar é código *MATLAB*.

As metáforas visuais *UMAT* (secção 3.7.2) não foi implementada no protótipo. Contudo esta está disponível no módulo *Matrix View* [21].

O desenho da base de dados proporcionou a identificação de possíveis relações entre os dados que são passíveis de análise, como por exemplo, as *toolboxes* e *m-files* têm uma estrutura hierárquica entre si bem como as regiões e neurónios no modelo SOM (subsecção 3.1.1). Dois casos de estudo são apresentados de seguida:



- Tendo em conta o estudo dos *concerns* o *TreeMap* pode ser utilizado para visualizar as densidades de cada *concern* presente no repositório, analisar que *toolboxes* contém um maior numero de *m-files* com instâncias *tokens* de um *concern* e encontrar os *m-files* mais significativos (com mais instâncias *tokens* de um *concern*). Aliada à utilização do *TreeMap*, a visualização de código está presente para o utilizador poder visualizar o código de um *m-file*.
- Estudar regiões dum modelo, estudar os neurónios contidos nelas e os *m-files* associados aos neurónios tendo em conta as *toolboxes* é mais um bom caso de estudo para representar num *TreeMap*. Através do uso do *TreeMap* pode se visualizar quais as maiores regiões de um modelo, mais um vez, como referido no parágrafo anterior, verificar se existem *toolboxes* em que existe um elevado número de *m-files* o que pode apontar para alguma modularidade presente na mesma.

Para além das metáforas visuais estudadas outros tipos de representações são necessárias como ferramentas complementares às metáforas visuais principais implementadas.

As “Representações Tradicionais” na Figura 5.1 tendo em conta o sistema representam tabelas que serão utilizadas como módulo para representação das relações entre dados e como módulos de representação do resultado de consultas requisitadas pelo o utilizador. Sendo as tabelas soluções básicas de representação de dados, estas não são suficientes para representar a multi-dimensionalidade das conclusões produzidas. Dessa forma, a representação dos dados deve ser feitas através da conjugação de metáforas visuais “tradicionais” e específicas.

Um ponto fulcral no desenvolvimento de um sistema em que a visualização de dados é um dos aspectos fundamentais do trabalho desenvolvido é a disposição dos elementos e a estilização dos mesmos, como referido na secção 3.7.1. Através da utilização do *Bootstrap*, vistas devem ser construídas onde um subconjunto de componentes se relacionam entre si.

## 5.5 Modelo de Dados

A solução proposta tem de lidar com a diversidade e quantidade de dados relacionados com todo o repositório *MATLAB* (ou seja, tanto os dados do repositório como os dados gerados pelas ferramentas de análise), o que exige uma solução de armazenamento e consulta poderosa de dados, i.e., todos os dados são representados como tuplos agrupados em relações numa base de dados relacional.

Tendo por base o afirmado e o referido na secção 4.1.2, uma solução *MySQL* foi utilizada.

### 5.5.1 Decomposição de *m-files* em estruturas de dados

O modelo relacional representado na Figura 5.3 representa *toolboxes*, *m-files* e os conteúdos completos dos mesmos (incluindo comentários, embora presentemente estes não

sejam utilizados em nenhum tipo de análise).

Novas toolboxes e m-files podem ser introduzidos no sistema em que versões de m-files são consideradas na presença de um *mfile* com a mesma *toolbox* e o mesmo nome na base de dados. A entidade *mfiles* tem um campo *base* que é *NULL* se não for versão de um m-file já presente na base de dados. Caso exista uma versão do m-file submetido na base de dados a entidade *mfiles* possui um *id\_mfile* como valor do campo do *m-file* de qual é versão. Esta abordagem permite analisar a alteração/evolução de um m-file.

Cada *m-file* é por sua vez decomposto em linhas de comentários e linhas de código. As linhas de comentário são utilizadas na construção do *m-file* original para posterior representação na visualização de código.

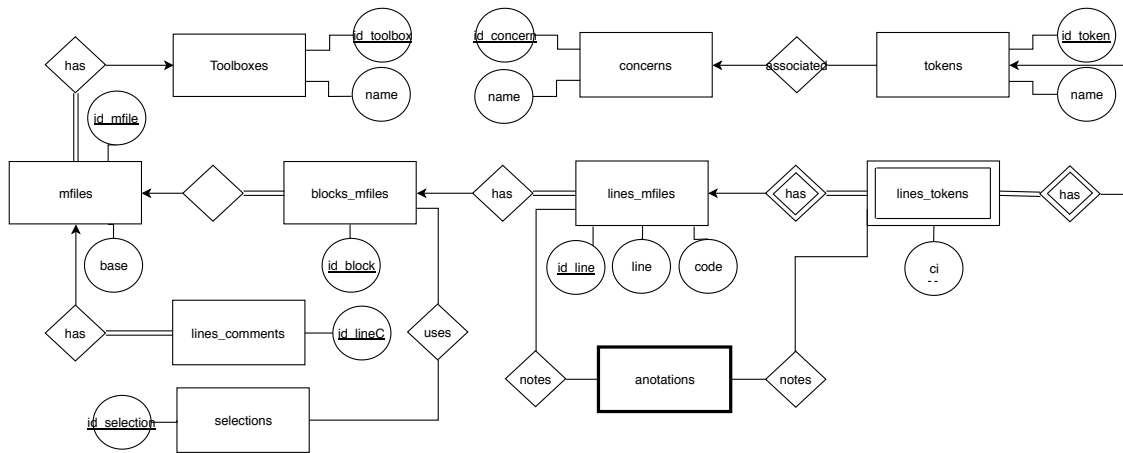


Figura 5.3: Diagrama ER da decomposição dos *m-files*

Os *tokens* (subsecção 2.4.2) são as estruturas de dados principais de análise. A Figura 5.3 representa as principais entidades e relações necessárias à modelação das relações entre *tokens* MATLAB presentes no repositório de acordo com a notação usada por Silberschatz et. al. [35]. Todos os *tokens* são armazenados no modelo: não só *word tokens* (subsecção 2.4.2), mas também *tokens* simbólicos, literais, etc.

Blocos (*block\_mfiles*) são as entidades intermédias que agrupam várias linhas de código e são parte de um m-file. Um m-file pode e normalmente é constituído por vários blocos. Nesta dissertação assume-se que cada m-file é constituído por apenas um bloco. Ou seja, todo o conteúdo de um m-file é considerado como um bloco.

Relações entre toolboxes, mfiles, blocos de código, linhas e instâncias *token* são simples relações *one-to-many*. Por exemplo, um tuplo *mfiles* pertence sempre a um tuplo *toolboxes* assim como um *block\_mfiles* pertence a um e um só m-file. Cada *block\_mfiles* tem um conjunto de *line\_mfiles*. Esta decomposição utilizou as funções básicas do parser fornecidas pela ferramenta CCCExplorer as quais serão apresentadas na sub-secção 6.2.2

Durante o *design* escolhemos modelar cada linha de código no repositório como um tuplo da entidade *lines\_mfiles* identificado pelo seu código *id\_line* único. No entanto, o número de linha da mesma no m-file e o *id\_mfile* são também identificadores únicos para cada linha. As linhas de comentários não fazem parte de *blocos* mas sim da entidade *mfiles*, como representado na Figura 5.3.

A presente abordagem *token-based* (ver subsecção 2.4.3) requer a identificação de cada instância *token*, funcionalidade já suportada pelo CCCExplorer. Cada linha de código é composta por uma sequência de instâncias *token* (ordem e a posição de cada ocorrência é importante).

Cada linha não vazia é composta pelo menos por uma instância *token* ou então é uma linha pura de comentário, *lines\_comments*. Visto que cada linha pode conter mais do que uma ocorrência da mesma instância *token*, uma entidade fraca (*lines\_tokens* na Figura 5.3) é usada como representação da mesma. A entidade fraca é codificada pelo *id\_line* da entidade *lines\_mfiles* e a posição da instância do *token* na linha (atributo *ic*). Dessa forma, é possível saber a linha que contém a instância *token* (atributo *line* é proveniente da entidade *lines\_mfiles*). Desta forma, uma relação indireta é criada em que cada instância *token* é associada ao seu código ou nome.

Um *token* pode ser associado a um determinado *concern* e um *concern* pode ser associado a vários *tokens* - representado pela entidade *belongs\_to* na Figura 5.3. O *design* tem em atenção que futuras análises podem vir a abordar o repositório de código baseado na seleção de diferentes critérios.

A entidade *selections* codifica conjuntos de blocos de código para serem estudados através de modelos SOM. As seleções dos blocos é realizada tendo em conta diferentes critérios. Para o modelo SOM estudado neste trabalho [26] são apresentados os critérios utilizados na codificação dos blocos de código (ver sub-secção 6.3).

A caixa *anotations* neste diagrama é apenas uma representação simplificada da relação entre *anotations* e os elementos que podem ter anotações, *lines\_mfiles* e *lines\_tokens*. Dessa forma, a noção *anotations* é abordada na secção 5.5.2 conjuntamente com a Figura 5.4 representativa do modelo relacional envolvido neste conceito.

### 5.5.2 Anotações sobre linhas de código e instâncias *tokens*

O modelo relacional representado na Figura 5.4 representa as *anotations* e as suas relações com as linhas de código (*lines\_mfiles*) e as instâncias *tokens* (*lines\_tokens*). Este novo modelo relacional apresentado especifica a abstração *anotations* representada na Figura 5.3.

A anotação de *tokens* é feita em relação a uma instância *token* de um m-file, ou seja, a anotação é feita sobre a instância *token*. Um *token* pode ter funcionalidades diferentes consoante o ambiente de programação em que está contido, por exemplo, na linguagem JavaScript o *token* '+' assume o papel de soma entre valores do tipo *inteiros/float/double*, enquanto entre *strings* funciona como concatenador das mesmas. A anotação por instância

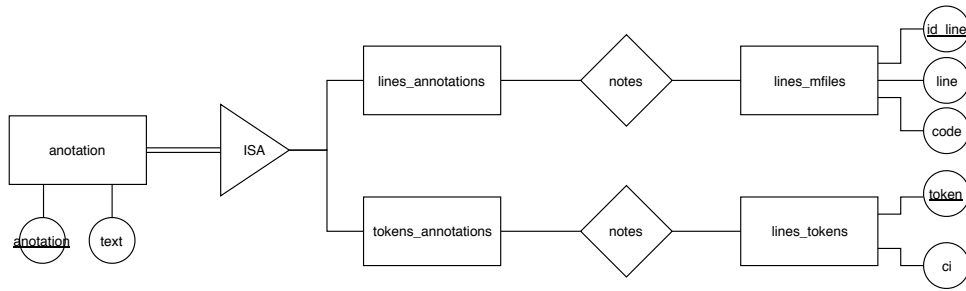


Figura 5.4: Diagrama ER referente ao sistema de Anotações

*token* (*tokens\_annotatons*) permite que ao mesmo *token* possam ser atribuídas anotações com caracterizações diferentes.

A anotação de linhas (*lines\_annotatons*) permite que linhas de código (*lines\_mfiles*) possam ser anotadas de forma a caracterizar o contexto de programação das instâncias *tokens*.

Uma *lines\_annotatons* tem o *id* do *user* que a criou, o *id* da *lines\_mfiles* correspondente e um campo com o texto da anotação. À semelhança das *lines\_annotatons*, as *tokens\_annotatons* tem o *id* da *lines\_token* em vez do *id* da *lines\_mfiles*.

### 5.5.3 Estruturas de dados a analisar e as suas relações no modelo de dados

O modelo relacional representado na Figura 5.5 modela a estrutura e os dados necessários à representação de modelos SOM (nem todos os elementos necessários à representação de modelos SOM são apresentados nesta sub-secção, ver também subsecção 5.5.4).

Uma *feature* é toda a propriedade pela qual seja possível estudar um dado. No âmbito do trabalho uma *feature* é um *concern*. Cada instância *token* presente na Tabela 2.2 é caracterizada por uma *feature* (*concern*).

As entidades *Sample* e *neuron\_vals* são caracterizadas por um valor para uma *feature*. A entidade *Sample* caracteriza um bloco de código em relação a uma *feature*, sendo o *value* a *Token Density* (secção 2.4.4) para essa *feature*. À semelhança, a entidade *neuron\_vals* representa o valor para uma *feature* de um neurónio (secção 3.1.1) utilizado num modelo SOM. O campo *value* na entidade *neuron\_vals* é inicializado com um valor aleatório no início do treino de um modelo SOM.

Para efeitos de análise através de modelos SOM tanto os dados a analisar, blocos de código, como os neurónios que fazem parte do treino de SOMs, são caracterizados no modelo de dados como *patterns*. Para treino de um modelo SOM um conjunto de *patterns* é codificado. Parte destes *patterns* dizem respeito a blocos de código que serão analisados, e a outra parte diz respeito aos neurónios utilizados no treino.

Um *pattern*, independentemente do que caracteriza é um vetor de valores. Quando um *pattern* diz respeito a um bloco de código, é um vetor de *Samples*, as quais participam de forma total na sua relação *in* com a entidade *Pattern*, representado na Figura 5.5. Quando um *pattern* diz respeito a um neurónio é um vetor de *neuron\_vals*.

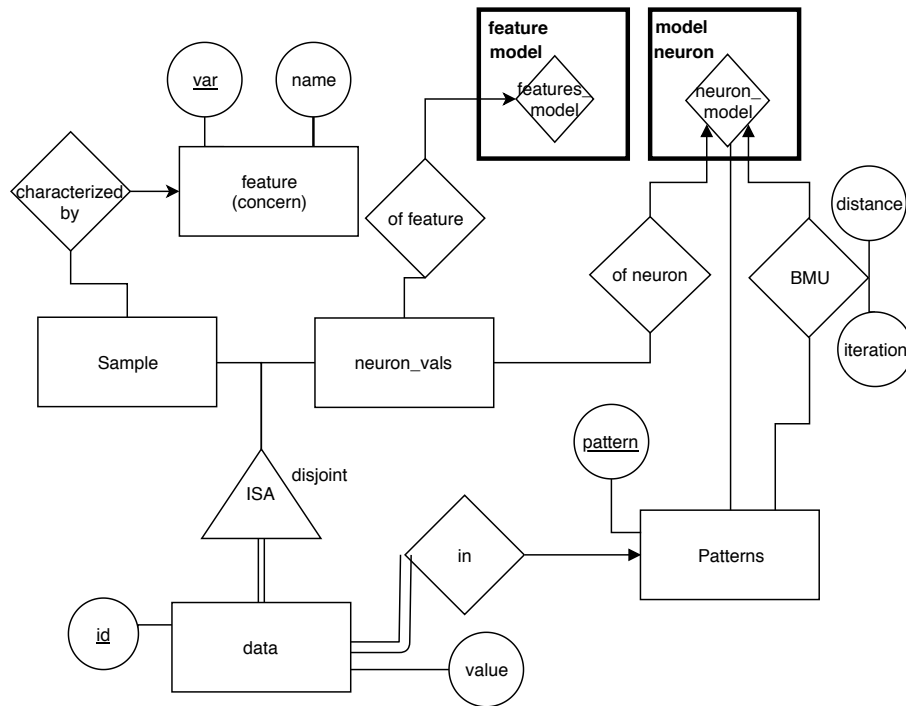


Figura 5.5: Diagrama ER referente ao armazenamento dos dados que alimentam as ferramentas de análise do sistema

Como referido na secção 3.1.1 a *BMU* é o neurónio (*neuron*) com o vetor de valores (conjunto de *neuron\_vals*) mais semelhante ao vetor de valores de um bloco de código (conjunto de *samples*). Tendo em conta o modelo de dados a entidade *BMU* representa os pares *patterns* (blocos-neurónios) codificados num modelo SOM. A entidade *BMU* é caracterizada pela a iteração do treino a que diz respeito e as distâncias euclidianas entre os pares *bloco-neurónio*.

O modelo de dados apresentado permite acomodar o conceito *streaming* abordado na secção (TRABALHOFUTURO). Tendo em conta a presente abordagem, cada novo m-file será um bloco caracterizado por um conjunto de *Samples* que poderá ser adicionado a um qualquer treino SOM, existente ou não. Após produção dos tuplos *samples* respeitantes aos novos blocos de código considerados para treino, as *BMUs* para esta nova iteração do treino são calculadas para cada *pattern* bloco de código considerado no modelo [26].

As caixas representadas por *feature\_model* e *model\_neuron* são abstrações representativas das relações que as entidades *neuron\_vals* e *patterns* presentes na Figura 5.5 possuem com entidades ainda não apresentadas que concluem a modelação de modelos SOM. As abstrações referidas são detalhadas na próxima sub-secção 5.5.4.

#### 5.5.4 Modelo de dados como suporte ao uso do SOM

O desenho lógico do sistema de suporte ao processo de treino e término de um modelo SOM está representado na Figura 5.6.

Um modelo SOM, como referido na subsecção 5.5.3, é caracterizado por um conjunto de *features*, um conjunto de neurónios e um conjunto de blocos de código. Assim, os *neuron\_vals* utilizados num modelo é restrito pelo próprio modelo e pelo conjunto de *features* e *neurons* utilizados.

Um modelo SOM consiste num conjunto de unidades onde cada unidade pode ser vista como a generalização da representação de um conjunto de *m-files* com valores de métricas semelhantes - denominadas também como protótipos e representados no modelo relacional apresentado na Figura 5.6 pela entidade *neurons*.

Para criação de um modelo SOM um conjunto de neurónios é selecionado para ser utilizado no treino do mesmo (secção 3.1.1). Dessa forma o conjunto de *neuron\_vals* a utilizar no treino é restrito pelo o modelo em questão e pelo sub-conjunto de neurónios selecionado. Para um modelo diferente o sub-conjunto de neurónios utilizado poderá ser diferente de outros modelos mas ao mesmo tempo conter neurónios que já foram utilizados em treinos diferentes. De forma a que o sistema permita esta particularidade a relação entre *model*, *neuron* e *neuron\_vals* têm de ocorrer sobre uma agregação.

As *features* são propriedades que limitam de igual forma os *neuron\_vals* considerados no modelo SOM. Apenas um sub-conjunto dos *neuron\_vals* codificado pela a agregação entre *model/neuron\_vals/neuron* será utilizado, visto que só os *neuron\_vals* que tiverem associados às *features* selecionadas serão utilizados.

Estas duas agregações visam representar que diferentes modelos podem incluir *neuron\_vals* e *features* que já foram utilizados em modelos anteriores, tudo depende do sub-conjunto de *neurons* e *features* selecionados.

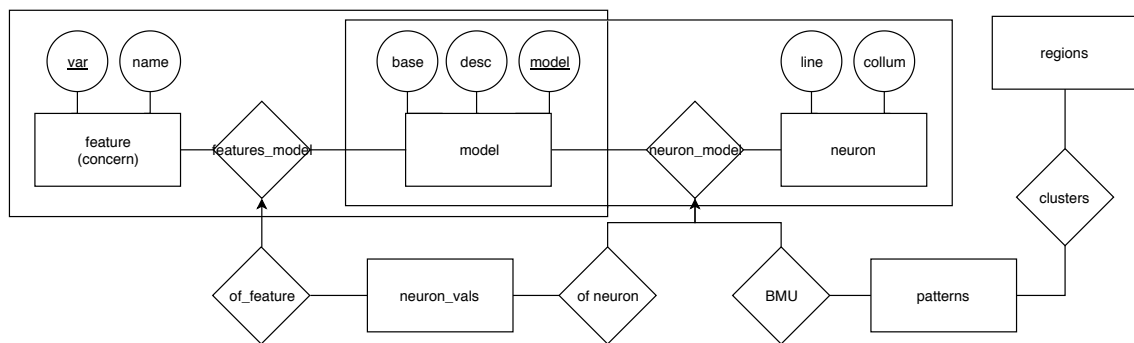


Figura 5.6: Diagrama ER relativo às várias instâncias SOM e estruturas inerentes ao treino do SOM

O componente *UbiSOM* realiza uma análise contínua do *stream* de dados e mantém um SOM que resume todos os *m-files* presentes no repositório, atualizando-o quando novos conteúdos são adicionados. O modelo relacional consegue lidar com múltiplas instâncias SOM, as quais estão representadas no modelo através do atributo *base* da entidade *model*, ver Figura 5.6. Isto permite abrir um caminho para utilizadores (devidamente treinados), para que estes possam especificar *queries* sobre modelos SOM (recorrendo à linguagem SQL). Novamente, embora o modelo de dados esteja preparado para esta funcionalidade

a mesma não foi implementada.

Vários neurónios são também agregados em regiões (secção 3.1.1) de unidades similares no SOM, esta informação é armazenada como tuplos na base de dados (entidade *regions*). Como referido na subsecção 3.1.2, m-files que tenham valores *Token Density* semelhantes nas *features* (*concenrs*) consideradas ficam associados ou à mesma *BMU* ou em neurónios com distância euclidiana entre eles mínima, formando-se assim *clusters* de neurónios com propriedades semelhantes. Como representado na Figura 5.6 *clusters* de *patterns* levam à formação de *regions*. A consideração das *regions* no sistema irá permitir através do uso da metáfora visual *TreeMap* uma análise sobre o algoritmo de *clustering* utilizado no treino de modelos SOM (ver sub-secção 6.5).





## IMPLEMENTAÇÃO E VALIDAÇÃO COM METÁFORAS VISUAIS

O presente capítulo tem como objetivo detalhar os aspetos relevantes da implementação das componentes, apresentar alguns detalhes da navegação do protótipo e apresentar resultados como forma de validação do trabalho realizado. O capítulo começa por abordar na secção 6.1 as tabelas que são utilizadas nos componentes apresentados no capítulo. Na secção 6.2 as alterações realizadas à ferramenta *CCCExplorer* para alimentar o modelo de dados da base de dados são apresentadas. Na secção 6.3 a transformação do modelo SOM estudado no artigo [26] para ser integrado no sistema, é apresentada. Na subsecção 6.4.1 a vista de código é apresentada e na secção 6.5 são abordados dois *TreeMaps* e análise ao modelo relacional do modelo SOM estudado [26]. A vista módulo *queries* é apresentada na secção 6.6 com os vários tipos de pesquisa disponíveis aos utilizadores *Joes*.

### 6.1 Implementação do Modelo de Dados em SQL

Ao longo deste capítulo *queries SQL* são utilizadas como ferramentas de análise sobre a base de dados. Na Listagem 6.1 estão representadas as instruções *SQL* da criação das tabelas, com os principais atributos, chave primária e chaves estrangeiras da tabela. Estas tabelas implementam o modelo de dados descrito na secção 5.5.

As tabelas *regions*, *toolboxes*, *mfiles*, *block\_mfiles*, *lines\_mfiles*, *lines\_tokens*, *tokens*, *concerns* e *patterns* correspondem respectivamente às entidades *regions*, *toolboxes*, *mfiles*, *block\_mfiles*, *lines\_mfiles*, *lines\_tokens*, *tokens*, *concerns* e *patterns*. As tabelas *clusters* e *bmu* correspondem respectivamente às relações *clusters* e *BMU* (ver secção 5.5).

Listagem 6.1: Instruções SQL utilizadas na construção das estruturas de dados para os *TreeMaps*

<pre> 1 CREATE TABLE 'toolboxes' ( 2   'id' int(10) UNSIGNED NOT NULL, 3   'name' varchar(300) NOT NULL, 4   PRIMARY KEY('id')); 5 6 CREATE TABLE 'mfiles' ( 7   'id' int(10) UNSIGNED NOT NULL, 8   'id_toolbox' int(10) UNSIGNED NOT NULL, 9   'name' varchar(100) NOT NULL, 10  PRIMARY KEY('id'), 11  FOREIGN KEY('id_toolbox')); 12 13 CREATE TABLE 'versions_mfiles' ( 14   'id' int(10) UNSIGNED NOT NULL, 15   'id_mfile' int(10) UNSIGNED NOT NULL, 16   PRIMARY KEY('id'), 17   FOREIGN KEY('id_mfile')); 18 19 CREATE TABLE blocks_mfiles ( 20   id int(10) UNSIGNED NOT NULL, 21   id_vers_mfile int(10) UNSIGNED NOT NULL, 22   PRIMARY KEY('id'), 23   FOREIGN KEY('id_vers_mfile')); 24 25 CREATE TABLE 'regions' ( 26   'id' INTEGER AUTOINCREMENT, 27   'name' varchar(300) NOT NULL, 28   PRIMARY KEY('id')); 29 30 CREATE TABLE clusters ( 31   'id' INTEGER, 32   'pattern' int(10) UNSIGNED NOT NULL, 33   'model' varchar(100) UNSIGNED NOT NULL 34   'region' int(10) NOT UNSIGNED NULL 35   PRIMARY KEY('id')); </pre>	<pre>   CREATE TABLE 'lines_mfiles' (     'id' int(10) UNSIGNED NOT NULL,     'id_block_mfile' int(10) UNSIGNED NOT NULL,     PRIMARY KEY('id'),     FOREIGN KEY('id_block_mfile'));     CREATE TABLE 'lines_tokens' (     'id' int(10) UNSIGNED NOT NULL,     'id_line' int(10) UNSIGNED NOT NULL,     'id_token' int(10) UNSIGNED NOT NULL,     PRIMARY KEY('id'),     FOREIGN KEY('id_line', 'id_token'));     CREATE TABLE 'tokens' (     'id' int(10) UNSIGNED NOT NULL,     'id_concern' int(10) UNSIGNED NOT NULL,     PRIMARY KEY('id'),     FOREIGN KEY('id_concern'));     CREATE TABLE 'concerns' (     'id' int(10) UNSIGNED NOT NULL,     'name' varchar(255) NOT NULL     PRIMARY KEY('id'));     CREATE TABLE 'patterns' (     'pattern' int(10) UNSIGNED NOT NULL,     PRIMARY KEY('pattern'));     CREATE TABLE 'bmu'(     'id' int(10) UNSIGNED NOT NULL,     'id_version_mfile' int(10) UNSIGNED NOT NULL,     'pattern' int(10) UNSIGNED NOT NULL,     PRIMARY KEY('id'),     FOREIGN KEY('id_version_mfile', 'pattern')); </pre>
--	---

O CCCExplorer cria *output SQL* para alimentar as tabelas *toolboxes*, *mfiles*, *block\_mfiles*, *lines\_mfiles*, *lines\_tokens* e *tokens* na base de dados (6.2). Para a representação do código *MATLAB* no protótipo, são utilizadas as tabelas *mfiles*, *block\_mfiles*, *lines\_mfiles*, *lines\_tokens* (6.4.1). As tabelas *toolboxes*, *mfiles*, *block\_mfiles*, *lines\_mfiles*, *lines\_tokens*, *tokens* e *concerns* são utilizadas para construção do *TreeMap* representativo do repositório de código (subsecção 6.5.1) e as tabelas *clusters*, *patterns*, *bmu*, *mfiles* e *toolboxes* são utilizadas para construção do *TreeMap* (subsecção 6.5.2) representativo das regiões identificadas do modelo SOM estudado [26]. No módulo de *queries* as tabelas *toolboxes*, *mfiles*, *block\_mfiles*, *lines\_mfiles*, *lines\_tokens* e *tokens* são utilizadas para realizar pesquisas na base de dados.

## 6.2 CCCExplorer

Na secção 5.2 foram indicadas as várias tarefas que o *CCCExplorer* tem de realizar para transformar um repositório *MATLAB* em dados que alimentem a base de dados. Ao longo desta secção o processo desta transformação bem como a implementação por detrás do processo são abordados.

### 6.2.1 Alterações realizadas na ferramenta

O *CCCExplorer* enquanto interpretador léxico já realizava a segmentação do código *MATLAB* em linhas e instâncias *tokens* em objetos *Java* que posteriormente eram representados num *output txt* com as suas características, como referido na secção 5.2. O necessário foi aproveitar a forma como a análise já era feita a cada m-file e produzir um *output* diferente ao *output* normal da ferramenta.

- *MainProdutorSQL.java*: consiste numa *main* similar à *MainGenBlocks.java* abordada na secção 2.5, no entanto cria diferentes ficheiros *output*, ficheiros *SQL*, para cada uma das estruturas necessárias a importar pelo *CCCExplorer*.
- *TokensDB.java*: consiste num objeto representativo de um *token* vindo da base de dados, possui 3 campos, sendo eles o *id* identificativo do mesmo na base de dados, o *id* do *concern* a que este pertence e ainda o seu nome. Este objeto possui ainda métodos para retorno de cada campo.

E os principais ficheiros alterados foram:

- *Token.java* devido à necessidade de considerar a linha em que a instância *token* foi encontrada bem como a posição da instância, *coluna de inicio e de fim* da mesma para a criação de um identificador único.
- *LexicalAnalyser.java* para criar o *output* das *linhas de comentários* e das *linhas sem comentários* do m-file. Esta classe teve de ser ainda alterada para que os novos campos criados no objeto instância *token* fossem contemplados no momento de análise dos mesmos numa linha.

A consideração da posição das instâncias *tokens*, *coluna inicial e final*, são o par de atributos que permitirá a análise do código *MATLAB* segundo o uso das *queries* sequenciais, modalidade específica de análise do módulo de queries. Na secção 6.6.1 este tópico será abordado.

### 6.2.2 Importação das estruturas de dados estudadas

De forma a popular a base de dados é necessário duas iterações do *CCCExplorer* sobre o repositório de dados.

Numa primeira abordagem apenas os *tokens* identificados na tabela 2.2 foram considerados. No entanto, de forma a potenciar a análise exploratória de dados, concluiu-se que era necessário que todos os *tokens* ( neste caso assumindo o papel de expressão atômica ) presentes no repositório fossem considerados. Assim levantou-se um problema, como saber todos os *tokens* existentes no repositório. A solução pensada foi, correr o *CCCExplorer* duas vezes.

- Na primeira iteração, o *CCCExplorer* é utilizado de forma a identificar todos os *tokens* ( neste caso assumindo o papel de expressão atômica ) existentes no repositório. Por cada *token* identificado no repositório é criado um objeto *Token*, ao qual é associado o *name* e o *id* do *concern* a que este pertence. Durante a análise realizada pelo *parser* do *CCCExplorer* a cada *token* de cada *linha*, é feita uma procura do *token* numa estrutura de dados que contém os *tokens* da tabela 2.2. Caso este esteja contido na estrutura, o objeto que o constitui contém informação sobre o *id* do *token* bem como o *id* do *concern*. Caso este *token* não esteja contido na estrutura, o *concern* é associado tendo em conta o elemento léxico que é.

Os dados dos objetos *Token* são transformados em ficheiros *SQL*, que posteriormente são importados na base de dados de forma a adquirirem um *id*.

A partir deste momento a base de dados contém todos os *tokens* presentes no repositório. A base de dados é agora exportada com *id's* de forma a que na segunda iteração quando cada instância *token* é identificada nos *m-files*, estas fiquem identificadas com o *id* da base de dados identificativo deste *token*.

Como é possível observar na Figura 6.2, existe mais um campo considerado, para além dos já referidos, na inserção de um *token*, o campo *description*, que corresponde a uma pequena descrição acerca do *token*, no entanto o seu preenchimento pode ser feito posteriormente.

Contrário ao que é normal no *SQL Standard* para uma instrução de inserção de dados, nas instruções representadas, utiliza-se a possibilidade do *SQL* de adicionar um *IGNORE* após o *INSERT*. Esta alteração possibilita o papel de a instrução não seja bem executada o importador da base de dados ignore essa instrução e prossiga a importação do restante *script*. No nosso caso, *queries* dão erro devido a *tokens* repetidos. Na identificação de todos os *tokens* presentes no repositório é normal que exista a identificação de *tokens* repetidos, e por isso mesmo, estes casos têm de ser ignorados.

Listagem 6.2: Código *SQL* para inserção de *tokens* na base de dados

```

1 INSERT IGNORE INTO tokens(id_concern , name, description) VALUES("1",
2 "nargin", " ");
3 INSERT IGNORE INTO tokens(id_concern , name, description) VALUES("2",
4 "fi", " ");
5 INSERT IGNORE INTO tokens(id_concern , name, description) VALUES("3",
6 "isfield", " ");
7 ...

```

- Na segunda interação, o *CCCExplorer* irá produzir ficheiros SQL para cada uma das estruturas estudadas na sub-secção 5.5.1. A produção destes ficheiros não poderia ser feita sem a primeira interação do *CCCExplorer*, pois o modelo de dados *lines\_tokens* necessita de saber o *id* do *token* identificado. Ou seja, nesta nova interação, aquando da identificação de cada *token*, uma procura deste é feita numa estrutura de dados que contém todos os *tokens* e respectivos *id* dos *concerns* associados. É nesta interação que código SQL para popular a base de dados de dados sobre as *toolboxes* de cada *m-file* e sobre eles mesmo é produzido.

Na segunda interação, os ficheiros produzidos e o seu conteúdo são:

- Um ficheiro que diz respeito apenas ao modelo de dados *toolboxes*:

Listagem 6.3: Código SQL para inserção de *toolboxes* na base de dados

```
1  INSERT IGNORE INTO toolboxes(id, name) VALUES("1", "/SourceForge-MATLAB-
2  Mfiles8/spm5-2011-12-14/spm5");
3  ...
```

- Um ficheiro sobre os *m-files* que contém *queries* para inserção dos dados em três modelos de dados diferentes, *mfiles*, *original\_mfiles* e *versions\_mfiles*:

Listagem 6.4: Inserção na tabela *mfiles* *original\_mfiles* e *version\_mfiles*

```
1  INSERT INTO mfiles(id, id_toolbox, name) VALUES("1", "cpxdotprod3.m");
2  INSERT INTO original_mfiles(id, id_mfile) VALUES("1", "1");
3  INSERT INTO version_mfiles(id, id_mfile, id_orig_mfile) VALUES(
4  "1", "1", "1");
5  ...
```

- Um ficheiro que contém informação sobre as linhas de comentários, modelo de dados *lines\_comments*, e linhas não vazias, *lines\_mfiles*.

Listagem 6.5: Código SQL para inserção de linhas de código e comentário na base de dados

```
1  INSERT INTO lines_mfiles(id, id_block_mfile, line, code) VALUES(
2  "1", "1", "1", " function [Creal, Cimag] = cpxdotprod3(Areal, Aimag,
3  Breal, Bimag) ");
4  INSERT INTO lines_comments(id, id_version_mfile, line, code) VALUES(
5  "1", "1", "2", " % complex dot product ");
6  ...
```

- Um ficheiro que contém a informação sobre cada *token* identificado em cada linha, modelo de dados *lines\_tokens*.

Listagem 6.6: Código SQL para inserção instâncias *token* na base de dados

```

1  INSERT INTO lines\_tokens(id, id_line, id_token, ci, cf) VALUES(
2  "1", "1", "1", "0", "7");
3  ...

```

Após a importação dos ficheiros de SQL produzidos na segunda iteração do CCCExplorer, a base de dados possui toda a informação necessária para a visualização de código, uso do sistema de anotações bem como da visualização dos dados num formato *TreeMap*.

Após a produção das métricas *TokenDensities*, podem-se começar a produzir *inserts* para adicionar à base de dados a informação e anotações do modelo SOM sobre o repositório.

### 6.3 Representação do modelo SOM estudado na base de dados

Está fora do âmbito desta dissertação a construção de modelos SOM. No entanto, transformou-se o modelo SOM estudado [26] num modelo relacional.

Para definir um modelo SOM começamos por configurar a definição de um conjunto de propriedades. Foi utilizado um módulo *sqlLite* para criar e armazenar o modelo estudado na base de dados. Dessa forma o modelo produzido anteriormente [26] foi traduzido para SQL de acordo com o sistema de base de dados criado.

Foi num módulo *sqlLite* exterior ao protótipo *web*, desenvolvido pelo professor Nuno Marques, que as configurações iniciais do modelo foram realizadas. As propriedades configuradas foram:

- **Features:** No estudo realizado todas as *features*, *concerns*, foram utilizadas no treino do modelo. A lista completa dos *concerns* utilizados está representada na Tabela 2.2.

Como referido na sub-secção 5.5.3 existem três tipos de *patterns* envolvidos num modelo SOM. Foram criados *patterns* para os neurónios, *patterns* para cada m-file e *patterns* representativos dos pares *feature-value* de cada m-file incluído no estudo.

Os valores referentes às *features* para cada m-file foram calculados pelo CCCExplorer no início do estudo [26].

- **Tamanho da rede neuronal:** Número de neurónios que serão treinados e qual a sua distribuição, um neurónio por cada abcissa e ordenada. O estudo considera uma rede neuronal constituída por 800 neurónios, 40 neurónios do eixo das abcissas, ou de comprimento da rede e 20 neurónios de ordenada, ou de largura da rede.

Os valores de cada par *feature-value* foram importados.

- **M-files:** Em trabalho anterior [26], realizaram-se testes de forma a procurar as melhores propriedades a considerar na análise dos m-files em SOM tendo por base a abordagem *Token-Based*. As restrições aplicadas sobre o repositório de análise

constituem um sub-conjunto de m-files o qual é modelado na base de dados pela entidade *selections* (ver sub-secção 5.5.1).

Após um processo de filtragem de dados positivos para análise realizado no trabalho anterior [26] definiram-se os seguintes parâmetros para seleção do subconjunto de m-files a utilizar. Dessa forma um m-file foi considerado caso:

- fosse composto pelo menos por 5 linhas de código (5 *lines\_mfiles*), ou seja  $LoC > 5$ ;
- contivesse 3 instâncias *tokens* (3 *lines\_tokens*) em que cada uma pertencia a um *concern* diferente da Tabela 2.2.

A tabela *block\_mfiles* é utilizada para se conseguir identificar qual o m-file que está a ser analisado.

Após restringir o repositório importado na base de dados segundo os parâmetros apresentados o conjunto de m-files codificados para o treino foram 3546 m-files.

Para além disso foram identificadas regiões na *UMAT* do modelo *SOM* em estudo, ver Figura 3.2.

As *BMUs* de cada m-file analisado foram guardadas. As regiões identificadas e abordadas no trabalho [26] foram criadas em base de dados bem como os *clusters* que as constituem. Para efeitos de estudo foram considerados os neurónios mais significativos (neurónios que se encontram no centro da região) de cada região. Idealmente deveria ter sido utilizado o algoritmo *flood* [23] para a identificação dos neurónios pertencentes às regiões. Com a importação do modelo em estudo poderemos analisar o mesmo através da utilização de um *TreeMap* bem como através de *queries* (ver sub-secção 6.5 e sub-secção 6.6 respetivamente).

## 6.4 Componentes das Metáforas Visuais

Para implementação das metáforas visuais recorreu-se ao uso da biblioteca *Bootstrap* e *ZingChart*. O *Bootstrap* foi utilizado na estilização dos componentes desenvolvidos.

O *frontend* desenvolvido consiste num conjunto de páginas que são compostas pelo conjunto de componentes desenvolvidos nesta dissertação. O protótipo é constituído por 3 páginas *web*. Cada uma destas páginas é caracterizada por um componente principal, sendo eles o Componente de Visualização de Código, o Componente *TreeMap* e o Componente Módulo de Queries. Todas as páginas possuem visualização de código e cada uma das páginas tem componentes auxiliares dedicados a cada um dos componentes principais.

De forma a evitar o load recursivo das páginas aquando da necessidade de montar os mesmos módulos mas com conteúdo diferente, os pedidos realizados sem ser o do

*load* inicial são realizados sob uma política ajax. Mais, os módulos desenvolvidos são sempre componentes dinâmicos. Ou seja, qualquer componente não existe no carregamento inicial da página, só posteriormente estes são montados.

Todas as páginas apresentadas têm um módulo de visualização de código, como pode ser visto nas Figuras representativas de cada página 6.1, 6.7 e 6.13.

Os módulos foram desenvolvidos de forma a poderem funcionar de forma auto-suficiente. Ou seja, existem módulos que dependem inevitavelmente de outros, contudo estes foram desenhados de forma a que possam ser construídas páginas *web* com um sub-conjunto destes módulos, podendo assim construir páginas através da adição e remoção dos módulos implementados.

### 6.4.1 Vista de Código

A página de visualização de código é constituída pelo componente de visualização de m-files e componentes auxiliares. Nesta página específica existe ainda disponível o modo de visualização comparativa de m-files. Como componentes auxiliares, a visualização de código tem um sistema de anotações e uma tabela com os m-files que pertencem ao mesmo neurónio do m-file que está a ser visualizado.

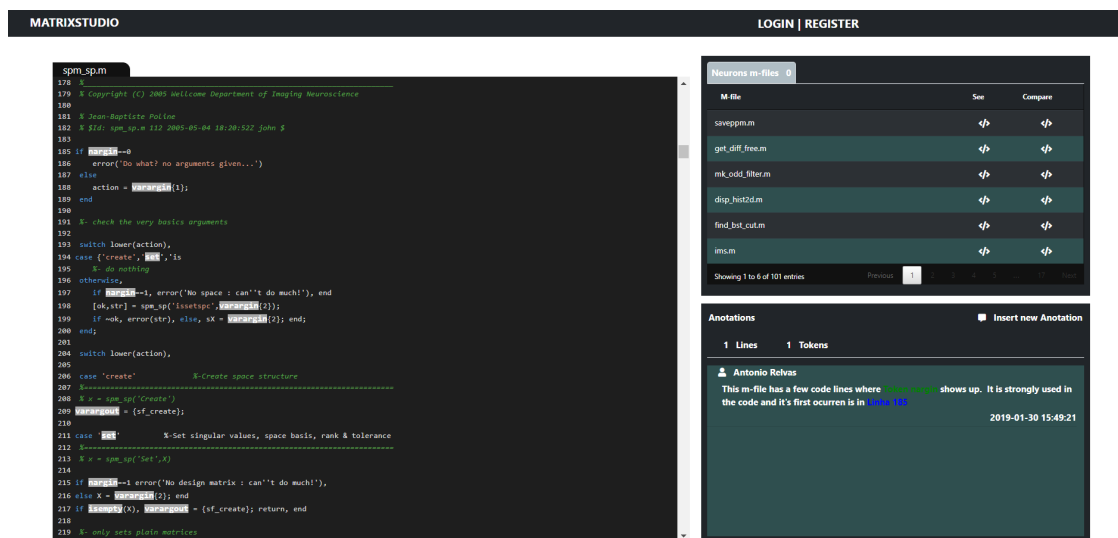


Figura 6.1: Vista de Visualização de Código

Para montar a visualização de código é necessário criar a sequência de linhas que constituem o m-file e posteriormente representar a sequência na página *web*. Um m-file é constituído por *lines\_comments* e um *block\_mfile*, ao qual pertencem *lines\_mfiles*, tal como descrito na secção 5.5 e representado na Figura 5.3.

A visualização de código é montada através da ordenação dos números de linha das linhas de comentários e os números de linha das linhas de código (*lines\_mfiles*). As linhas em branco de um m-file não são tidas em conta na base de dados. Contudo, durante a



montagem do m-file, se na sequência resultante dos números de linhas existirem números de linha em falta, linhas em branco são adicionadas até que a sequência de número de linhas seja contíguo, mantendo assim a estrutura original do m-file.

Para além da representação original do m-file, o conteúdo do mesmo é analisado pela *API highlight.js*. A *API* estiliza visualmente o conteúdo do m-file consoante a semântica dos *tokens* que o constituem, se o *token* for uma *keyword* tem um determinado estilo associado, se fizer parte de um comentário terá outro estilo visual associado e o processo repete-se para cada elemento distinto de semântica da linguagem MATLAB. Na Figura 6.2 é possível observar os estilos diferentes associados aos vários elementos semânticos diferentes do código MATLAB após a análise realizada pela *API highlight.js*. A *keyword* 'function' está representada com uma cor de letra azul e todas as outras *keywords* são representadas com a mesma cor. Os comentários são representados com cor de letra verde. Código sem semântica própria associada, ou seja, nomes de funções criadas pelos utilizadores, variáveis, ou instâncias de *tokens* do tipo símbolo, são representadas com cor de letra branco, ou seja, é tratado como texto normal. No decorrer do processo de montagem do m-file na presença de instâncias *tokens* com um *concern* associado, são tidas em conta como casos especiais. Essas instâncias *tokens* são realçadas de forma diferente em relação ao restante código por meios de CSS. A instância *token* fica com uma cor de fundo cinzenta. Esta alteração não é efetuada pela *API* mas sim por adicionar a classe CSS *token* a estes elementos. Na primeira linha do m-file "spm\_sp.m" representado no módulo de visualização de código Figura 6.1, duas instâncias *tokens* com um *concern* associado (ver Tabela 2.2) que estão representadas com uma cor de fundo diferente (cinzenta) do restante código (preto).

## 6.4.2 Vista Comparativa

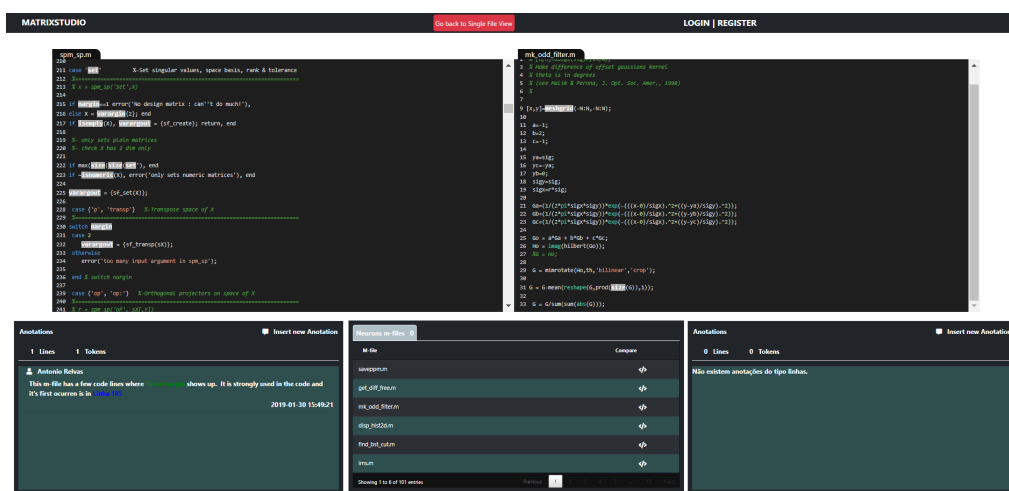
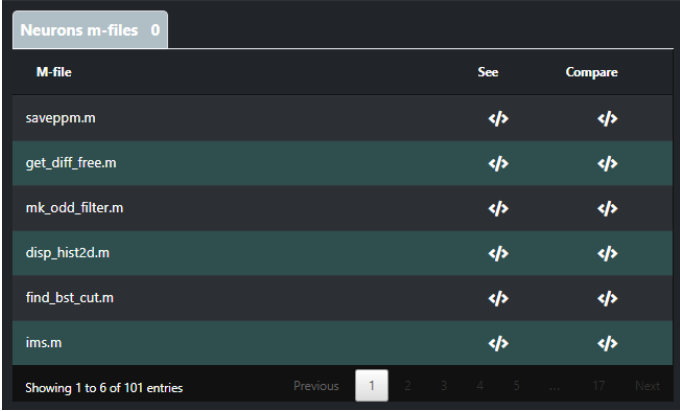


Figura 6.2: Vista Comparativa de Visualização de Código

A vista comparativa é uma extensão da vista de código. A vista é formada principalmente pela alteração das classes CSS dos módulos implementados. Esta vista permite a comparação da visualização entre m-files, representado na Figura 6.2.

### 6.4.3 Módulo m-files de Neurónio

O módulo m-files do Neurónio contém em si os m-files associados a um neurónio, permitindo a visualização dos mesmos e está representado na Figura 6.3, tendo como nome do módulo *Neurons m-files*. Este módulo funciona com os dados provenientes do modelo estudado [26]. Cada neurónio tem um conjunto de m-files associado.



The screenshot shows a web interface titled 'Neurons m-files 0'. It contains a table with three columns: 'M-file', 'See', and 'Compare'. The table lists six m-files: 'saveppm.m', 'get\_diff\_free.m', 'mk\_odd\_filter.m', 'disp\_hist2d.m', 'find\_bst\_cut.m', and 'ims.m'. Each row has a 'See' button with a magnifying glass icon and a 'Compare' button with a double arrow icon. At the bottom, there is a pagination bar showing 'Showing 1 to 6 of 101 entries' and a set of page numbers (1, 2, 3, 4, 5, ..., 17, Next) with 'Previous' and 'Next' links.

M-file	See	Compare
saveppm.m		
get_diff_free.m		
mk_odd_filter.m		
disp_hist2d.m		
find_bst_cut.m		
ims.m		

Showing 1 to 6 of 101 entries    Previous    1    2    3    4    5    ...    17    Next

Figura 6.3: Módulo m-files do Neurónio

O nome dos m-files é utilizado como propriedade representativa na tabela. Cada linha da tabela tem o nome do m-file e dois elementos clicáveis. Ao clicar no elemento da segunda coluna (*SEE*) um novo m-file é mostrado no visualizador de código, e ao clicar no elemento da terceira coluna (*COMPARE*) um novo visualizador de código é disposto ao lado do atual, ficando o código dos dois m-files lado a lado, ou seja, passamos a estar na vista Comparativa, Figura 6.2.

De forma a estilizar de uma forma simples e rápida este módulo tabela, a *API dataTables* foi utilizada. Após construção do *HTML* base de uma tabela, esse elemento é reformatado pela *API*. A utilização desta *API* foi para a criação da paginação dos resultados da tabela.

### 6.4.4 Módulo de Anotações

O módulo de anotações é composto por um módulo de inserção de dados Figura 6.4, um módulo de visualização de anotações Figura 6.5 identificado pelo nome *Anotations*, e uma extensão do módulo de visualização de anotações foi criado e associado ao código, Figura 6.6.

### Módulo de Inserir Anotações



Figura 6.4: Módulo Inserção de Anotações

O editor de texto é criado como um elemento *HTML textarea*. Após inicialização da *API Summercode* (ver secção 4.2.4) com esse elemento *textarea*, um componente *HTML* com essa *textarea* é montado. Este editor rico de texto tem funcionalidades básicas de formatação de texto. O uso desta *API* permite a introdução de uma *string* que é interpretada como linguagem *HTML*, funcionalidade necessária para suportar parte do funcionamento do módulo de anotações, mais especificamente o sistema de referenciamento.

O sistema de referenciamento, conjunto de botões dispostos do lado direito do editor de texto, foi criado de forma a que um utilizador enquanto faz uma anotação possa utilizar referencias a elementos do m-file durante a escrita normal da sua anotação. Estes elementos de referência têm um formato específico de id de elemento *HTML* e uma classe *CSS* distinta para ambos. Este é constituído pelos id's do m-file e da linha e/ou token.

Na submissão dos dados da anotação na base de dados, é analisada a *string* que compõe a anotação.

Caso na *string* exista a expressão '*class=*' assume-se que se está na presença de uma anotação. Caso a seguir na *string* venha *token\_refer* ou *line\_refer*, que representa que é uma anotação do tipo *token* e linha respetivamente, mais uma vez se assume que é parte de uma anotação. Tendo em conta que o conteúdo analisado é uma *string* e não guarda quaisquer tipos de características *HTML*, pode existir o caso em que tenha sido um utilizador a escrever certas palavras que são interpretadas como parte de uma anotação. Dessa forma não existe certeza de que é uma anotação. Só quando encontrando o tipo de notação abaixo representado se tem as propriedades necessárias para a criação de uma anotação na base de dados.

*mfile\_31424\_idLine\_1\_refer*  
*mfile\_31424\_idLine\_1\_idToken\_2555\_refer*

O conteúdo não respeitante a linguagem *HTML* e os id's exemplo acima referidos são os dados necessários à criação de uma anotação na base de dados.

## Módulo de Visualização de Anotações

No módulo de visualização existem duas tabs *Bootstrap* que permitem permutar entre ver as anotações realizadas a linhas do m-file que está a ser visualizado, ou às instâncias *tokens* associadas a um *concern* presentes no m-file.

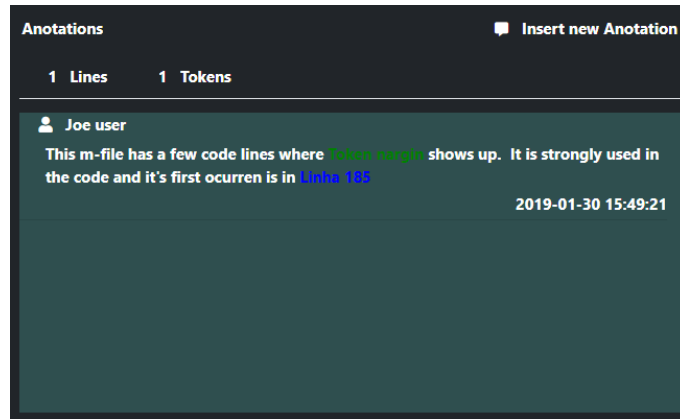


Figura 6.5: Módulo Visualização de Anotações

## Extensão do módulo de visualização de Anotações - Popover Annotation

Elementos *popovers*, abordados na seção 4.2.1, foram utilizados fundamentalmente para construir uma extensão às funcionalidades do módulo de anotações, apresentar informação sobre a instância *word token* que está a ser alvo do *hover* e a que *concern* a instância *word token* pertence.

- o *token*, pequena descrição do mesmo retirada do *MathWorks*;
- sobre o *concern* a que o *token* pertence, com uma breve descrição relativa ao *concern*;
- as anotações realizadas à instância *token* que está ser alvo de *hover*;

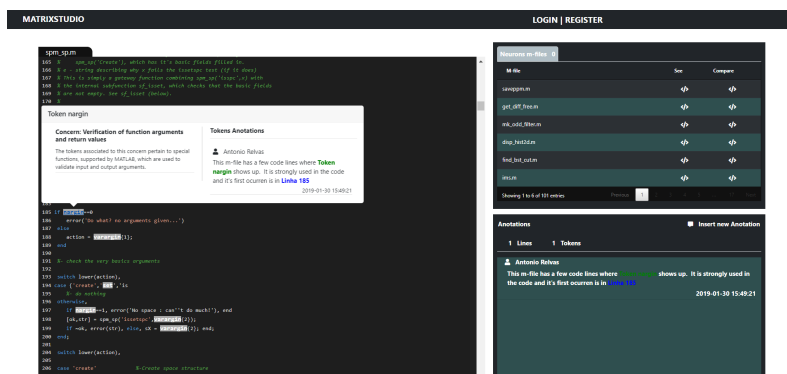


Figura 6.6: Popover Token

## 6.5 TreeMaps e análise do modelo relacional do modelo SOM

Existe uma página no protótipo que utiliza a biblioteca *JavaScript ZingChart* (ver subsecção 4.2.2) para representação global do repositório de código *MATLAB* (relações entre *Concerns* -> *Toolboxes* -> *M-files* considerando o número de instâncias *tokens* associadas a um *concern*) através da metáfora visual *TreeMap*. No primeiro *TreeMap* acima referido, a dimensão representada é o número de instâncias tokens de um concern por m-file, o que afeta diretamente a disposição dos elementos na representação e a área ocupada por cada elemento em cada nível. Já no caso do segundo *TreeMap*, a dimensão é o número de m-files, em que esta afeta apenas a área ocupada por cada elemento em cada nível.

Como representado na Figura 6.7 este tipo de vista do protótipo é constituída por um módulo de visualização de código e um módulo *TreeMap*. Na seleção de um m-file visualizado no *TreeMap* o conteúdo do mesmo é representado no módulo de visualização de código.

Os *TreeMaps* implementados no protótipo seguem uma representação balanceada dos dados. Este tipo de representação permite representar um maior número de elementos em cada nível da hierarquia. Quando testado segundo os outros tipos de disposição de dados, horizontal e vertical, o número de dados representados é muito pequeno. Os dados apenas são visíveis através do *hover* sobre os elementos constituintes do *TreeMap*.

As cores são utilizadas como fator diferenciador entre os *concerns* representados. Um evento *hover* está associado aos *TreeMaps* montados pelo *ZingChart* para que quando é realizado o *hover* sobre um elemento, consoante qual o elemento alvo do evento, o seu nome ou número de instâncias *token* é apresentado.

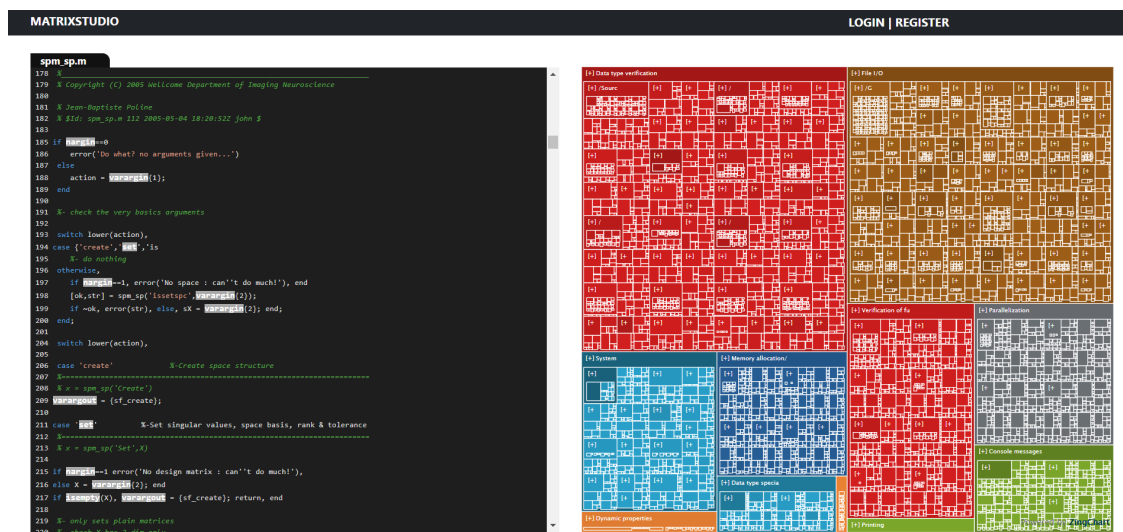


Figura 6.7: Vista TreeMap

### 6.5.1 Concerns - Toolboxes - M-files

A *query* representada na Listagem 6.7 é utilizada para criar a relação entre as várias entidades (ver secção 5.5 e no início do capítulo 6) que participam na relação de dados hierárquicas.

Listagem 6.7: *Query SQL* que cria a relação entre os elementos constituintes do *TreeMap*

```

1 SELECT
2   concerns.name as concern ,
3   toolboxes.name as toolbox ,
4   mfiles.name as mfile_name ,
5   count(tokens.id) as tokensPerConcern
6 FROM
7   toolboxes
8   INNER JOIN mfiles ON toolboxes.id = mfiles.id_toolbox
9   INNER JOIN versions_mfiles ON mfiles.id = versions_mfiles.id_mfile
10  INNER JOIN blocks_mfiles ON versions_mfiles.id = blocks_mfiles.id_vers_mfile
11  INNER JOIN lines_mfiles ON blocks_mfiles.id = lines_mfiles.id_block_mfile
12  INNER JOIN lines_tokens ON lines_mfiles.id = lines_tokens.id_line
13  INNER JOIN tokens ON lines_tokens.id_token = tokens.id
14  INNER JOIN concerns ON concerns.id = tokens.id_concern
15 WHERE
16   concerns.id Between 1 AND 11
17 GROUP BY
18   concern , toolbox , mfile_name

```

Na Listagem 6.7 pelo o corpo do *SELECT* da *query* podemos identificar os dados que serão representados no *TreeMap*, o nome dos *concerns*, o nome das *toolboxes*, o nome dos *mfiles* e a contagem de instâncias *tokens* de cada *concern* em cada *mfile*. O corpo do *FROM* representa a relação hierárquica existente entre os modelos de dados. A área ocupada por cada dado é representada pelo valor de do atributo “tokensPerConcern” visível no *SELECT* da Listagem 6.7.

Um *JSON* tem de ser construído com os dados obtidos da base de dados para que a biblioteca *ZingChart* possa criar o *TreeMap*. Parte da estrutura do *JSON* está representada na Listagem 6.8.

Listagem 6.8: Estrutura de dados *JSON* que alimenta a *API ZingChart* para representação do *TreeMap* em estudo

```

1 { "text": "Data type verification",
2   "children": [ { "text": " /Benchmarks-Mfiles/MatlabToCL Benchmarks ",
3     "children": [ {
4       "text": "dilate.m", "value": 2
5     }, {
6       "text": "matmul_nv.m", "value": 7
7     }, ...
8   }, { "text": "Console messages",
9     "children": [ { "text": " /Benchmarks-Mfiles/sd-vbs/benchmarks/localization/src/matlab ",
10      "children": [ {
11        "text": "generateSample.m", "value": 1
12      } ]
13    }, { "text": " /Benchmarks-Mfiles/sd-vbs/benchmarks/svm/src/matlab ",
14      "children": [ {

```



```

15     "text": "script_run_profile.m", "value": 2
16   }, {
17     "text": "selfCheck.m", "value": 2
18   }
19   ]}, ...}

```

A visualização inicial do *TreeMap* resultante do *JSON* 6.8 está representada na Figura 6.8. Nesta visualização estão representados os vários *concerns* identificados no estudo. A área ocupada por cada retângulo reflete o número de toolboxes em que foram identificados m-files com instâncias *tokens* do *concern*. Quanto maior o número de toolboxes, maior a área do retângulo. Este dado é corroborado pela Tabela 6.2, resultados obtidos por consulta à base de dados.

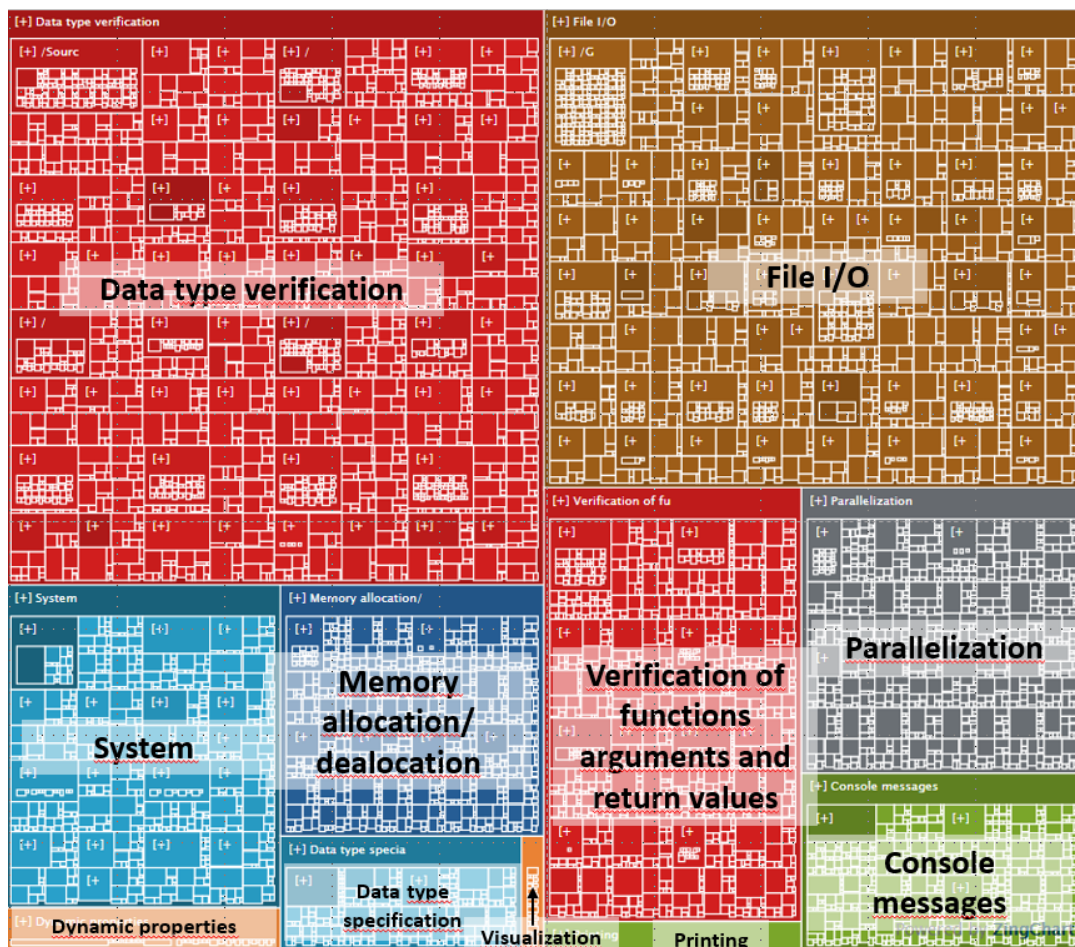


Figura 6.8: *TreeMap* representativo de todos os *concern* do repositório

No primeiro nível podemos observar a densidade de cada *concern* presente no repositório. Como referido na secção 3.8, o algoritmo do *TreeMap* tendo em conta o tipo de representação de balanceamento, dispõe os dados com maior densidade de uma propriedade, no nosso caso ocorrência de instâncias *tokens*, do topo esquerdo para o inferior direito. O *concern* “*Data type verification*” está disposto no canto superior esquerdo, o que significa que tendo em conta o atual repositório, existem mais m-files com instâncias

Concern	Nº de toolboxes
Data type verification	182204
File I/O	151618
Verification of function arguments and return values	65484
System	52035
Parallelization	47344
Memory allocation/deallocation	38692
Console messages	29321
Data type specialization	16104
Printing	4433
Visualization	1590

Tabela 6.1: Número de toolboxes identificadas com instâncias tokens, por concern

Concern	Toolbox	Nº de instâncias token
Data type verification	/SourceForge-MATLAB-Mfiles8/spm5-2011-12-14/spm5	6249
Data type verification	/SourceForge-MATLAB-Mfiles4/JAABAReleaseVer0.4.0/perframe	3388
Data type verification	/Code-Repositories-Mfiles/Classification/SOM MATLAB Toolbox/somtoolbox2_Mar_17_2005/somtoolbox	3073
Data type verification	/Code-Repositories-Mfiles/somtoolbox	3073
Data type verification	/SourceForge-MATLAB-Mfiles8/spm2/spm2	2738

Tabela 6.2: Toolboxes com maior número de instâncias tokens do concern Data type verification

*tokens* do *concern* referido.

Iremos a analisar o *concern* “Data type verification na visualização” por ser o *concern* que possui a maior densidade de instâncias *tokens* no repositório. Assim, uma *query* foi executada na base de dados para descobrir quais as toolboxes que continham mais instâncias *tokens* do *concern* “Data type verification” de forma a visualizar estes resultados no *TreeMap*. Na Tabela 6.2 os resultados estão apresentados e na Figura 6.9 está representada a parte superior esquerda do *TreeMap* codificado pela seleção do *concern*.

Considerando os resultados da Tabela 6.2, vamos analisar a visualização *TreeMap* e a informação que representa sobre os dados do repositório.

Listagem 6.9: Seleção do *concern* “Data type verification” no modelo de dados

```

1 ...WHERE
2 concerns.name = 'Data type verification'
3 GROUP BY
4 toolboxes , mfile_name

```

Clicando no nome do *concern*, a visualização do *TreeMap* é atualizada e o que é representado está na Figura 6.10 quando considerando o *concern* “Data type verification”. Esta visualização é representada pela seleção do conjunto de dados referentes ao *concern* a analisar. Esta seleção é efectuada no modelo de dados por refazer o campo “WHERE” da *query* 6.7 (ver Listagem 6.9).

Neste nível as toolboxes são representadas. Analisando as áreas dos retângulos verifica-se que estas são similares entre si, o que indica que o número de instâncias *tokens* por



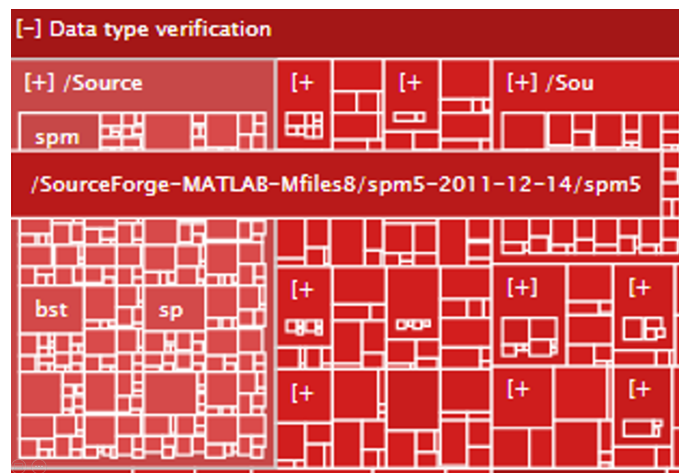


Figura 6.9: *TreeMap* ao nível do *concern* Data type verification

toolbox são semelhantes entre si. À exceção do caso da toolbox “*SourceForge-MATLAB-Mfiles8/spm5-2011-12-14/spm5*” que apresenta uma área superior às restantes. Esta toolbox foi codificada como a toolbox com maior número de instâncias *tokens* do *concern* no repositório (resultados da Tabela 6.2). Na visualização *TreeMap*, Figura 6.9, é o primeiro retângulo representado o que é caracterizado por ser o elemento que possui uma maior densidade da propriedade a ser avaliada.

A toolbox “*/SourceForge-MATLAB-Mfiles6/NBTalpha-RC3a/External/EGlab/external/biosig-partial/t200\_FileAccess*” representada na Figura 6.9 por uma cor mais intensa (canto inferior direito) possui um dos ficheiros mais significativos (maior número de instâncias *tokens* do *concern*, ver secção 5.4) do *concern* em análise. A intensidade da cor dos retângulos permitem que o utilizador encontre facilmente os m-files significativos.

Após feita a análise à visualização dos toolboxes no *TreeMap* clicamos sobre a toolbox “*SourceForge-MATLAB-Mfiles8/spm5-2011-12-14/spm5*” para acedermos à listagem de m-files que estão associados. Na Figura 6.10 está representada a visualização do conjunto de m-file associados à toolbox no *TreeMap*.

Esta visualização é representada pela seleção do conjunto de dados referentes à toolbox a analisar. Esta seleção é efectuada no modelo de dados por refazer o campo “*WHERE*” da *query* 6.7 (ver Listagem 6.10).

Listagem 6.10: Seleção da toolbox “*SourceForge-MATLAB-Mfiles8/spm5-2011-12-14/spm5*” no modelo de dados

```

1  ...WHERE
2    toolboxes.name = 'SourceForge-MATLAB-Mfiles8/spm5-2011-12-14/spm5'
3  GROUP BY
4    toolboxes, mfile_name

```

Neste nível é possível saber que m-files estão contidos na toolbox e quantas instâncias *tokens* tem cada m-file pertencente ao *concern*.

Através da API *ZingChart* e conjuntamente com a linguagem *JavaScript* programou-se

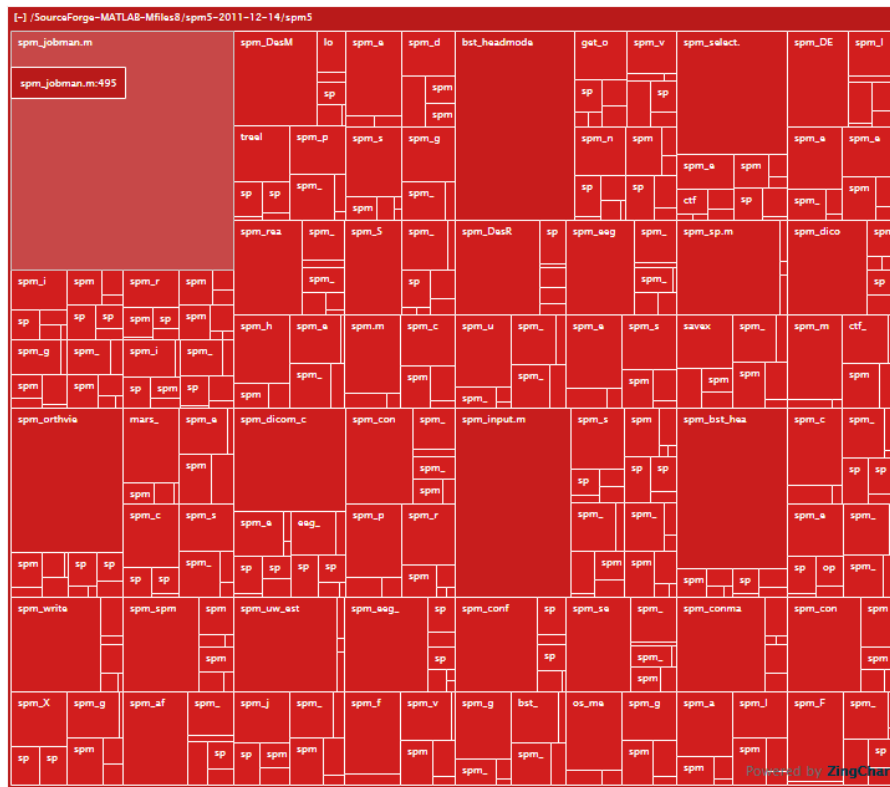
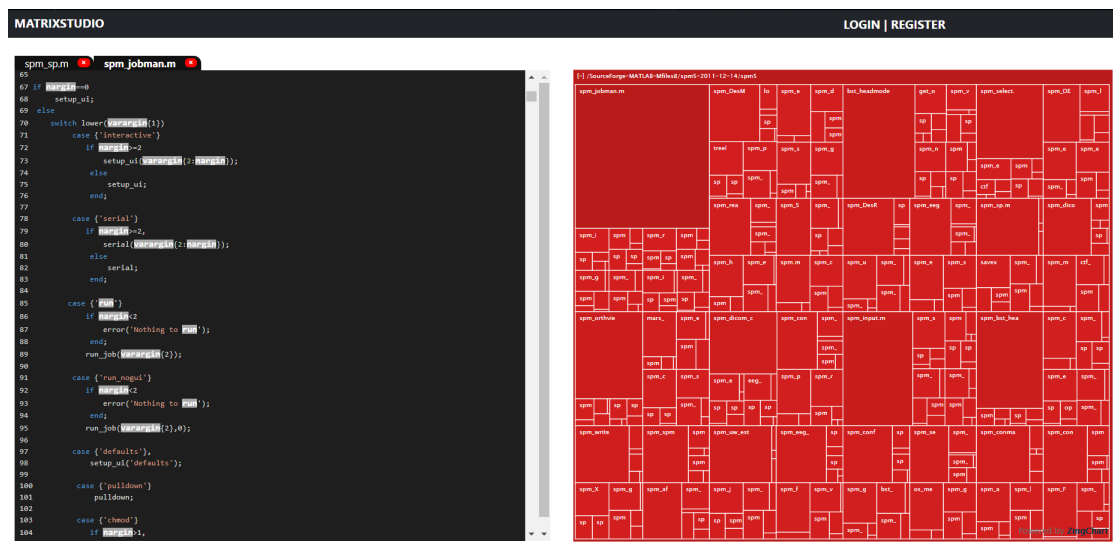


Figura 6.10: TreeMap ao nível m-file

um evento *click* sobre as representações dos m-files no *TreeMap*. Ao clicar sobre um m-file o seu código é representado no módulo de visualização de código da vista. Assim, clicando no m-file “spm\_jobman.m” a vista é atualizada para apresentar o conteúdo de código do mesmo, ver Figura 6.11. Nesta está representado o código do m-file selecionado.

Figura 6.11: Vista *TreeMap* com visualização do código do m-file selecionado no *TreeMap*

## 6.5. TREEMAPS E ANÁLISE DO MODELO RELACIONAL DO MODELO SOM

Região	Pattern	Toolboxes	M-files name
A1	101780	/SourceForge-MATLAB-Mfiles4/JAABAReleaseVer0.4.0/misc	drawellipse.m
A1	101780	/SourceForge-MATLAB-Mfiles5/matlab_hyperspectral_toolbox_v0.07	hyperOsp.m
...	...	...	...
A2	101520	/SourceForge-MATLAB-Mfiles7/POLGUI/polgui_ver4_r14	montage_modaldlg_r14.m
A2	101560	/SourceForge-MATLAB-Mfiles5/matGeom-1.1.6/matGeom/graphs	drawNodeLabels.m
...	...	...	...

Tabela 6.3: Formato CSV utilizado na construção dos *TreeMaps*

### 6.5.2 Regions - Patterns - M-files

A *query* representada na Listagem 6.11 é utilizada para criar a relação entre as várias entidades que (ver secção 5.5 e no início do capítulo 6) participam na relação de dados hierárquicas.

Listagem 6.11: *Query SQL* que cria a relação entre os elementos constituintes do *TreeMap*

```

1 SELECT
2     regions.name as region ,
3     patterns.pattern as neuron ,
4     toolboxes.name as toolbox ,
5     mfiles.name as mfile_name
6 FROM
7     regions
8     INNER JOIN clusters ON regions.id = clusters.region
9     INNER JOIN patterns ON clusters.pattern = patterns.pattern
10    INNER JOIN bmu ON patterns.pattern = bmu.pattern
11    INNER JOIN versions_mfiles ON bmu.id_version_mfile = versions_mfiles.id
12    INNER JOIN mfiles ON versions_mfiles.id_mfile = mfiles.id
13    INNER JOIN toolboxes ON mfiles.id_toolbox = toolboxes.id
14 WHERE
15     clusters.model = 100010
16 GROUP BY
17     region , neuron , toolbox , mfile_name

```

Na Listagem 6.11 pelo o corpo do *SELECT* da *query* podemos identificar os dados que serão representados no *TreeMap*, o nome das regiões, um inteiro identificativo do *pattern*, o nome das *toolboxes* e o nome dos m-files. O copro do *FROM* representa a relação hierárquica existente entre os modelos de dados. Na cláusula *WHERE* o modelo estudado [26] é identificado como *model 100010* no modelo relacional.

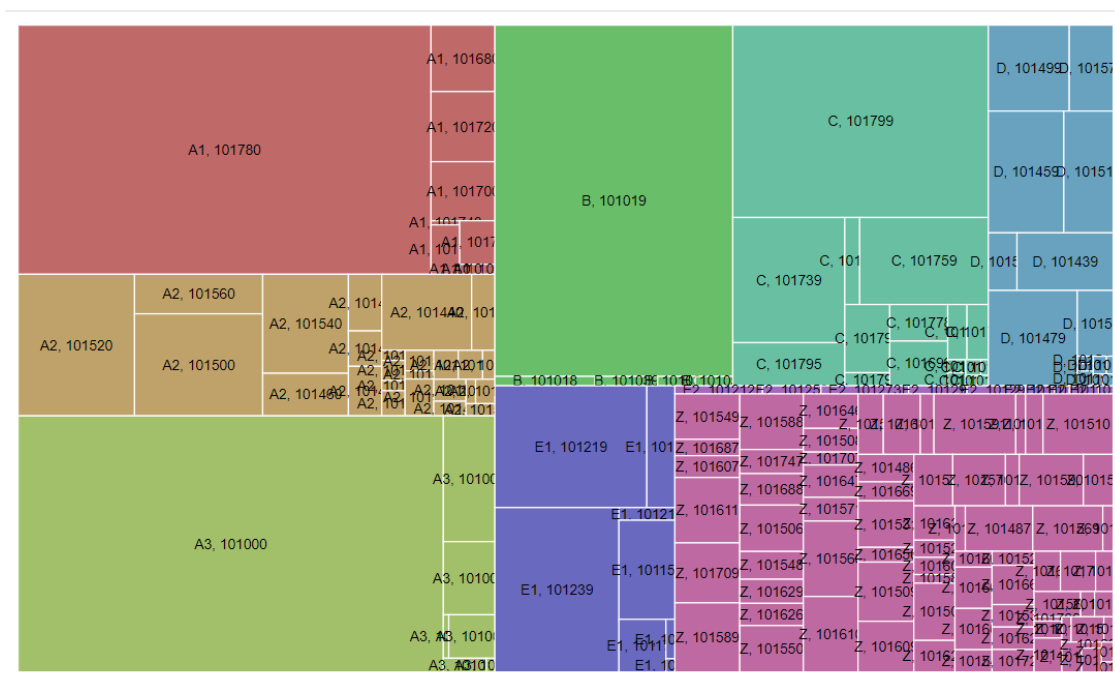
Durante a implementação destes *TreeMaps* não se conseguiu representar o resultado da *query* 6.11 através da ferramenta *ZingChart*. No entanto foi possível apresentar o *TreeMap* para esta mesma *query* recorrendo ao site *RAW*. As representações desta subsecção recorrem ao código *embed* para a criação de vários *TreeMaps* para representação visual das relações entre Regiões -> Neurónios -> Toolboxes -> M-files do modelo SOM em estudo [26].

A construção dos seguintes *TreeMaps* foram criados segundo ficheiros CSV com o conteúdo resultante da *query* 6.11. Este *TreeMap* dispõe as regiões por ordem alfabética ao contrário do *TreeMap* 6.5.1 já estudado em que a disposição é feita da esquerda para a direita consoante a densidade dos resultados. Pequena parte representativa do resultado

Região	Nº de m-files
A3	1036
Z	1034
A1	1008
C	782
B	728
A2	560
E1	437
D	382
E2	30

Tabela 6.4: Número de neurónios pertencentes a cada região do SOM estudado [26]

da *query* 6.11 é apresentado na Tabela 6.3.

Figura 6.12: *Treemap* das regiões do modelo SOM estudado [26]

Na Figura 6.12 está representado o *Treemap* representativo das regiões existentes no modelo estudado [26] bem como da proporção de m-files por região. Neste *Treemap* a label é o nome da região (referentes ao modelo SOM estudado [26]) e um identificador do neurónio na base de dados.

Listagem 6.12: *Query SQL* que procura os m-files de cada região

```

1 SELECT
2   regions.name as region ,
3   patterns.pattern as neuron ,
4   toolboxes.name as toolbox ,
5   mfiles.name as mfile_name
6 FROM
7   regions

```

```

8 INNER JOIN clusters ON regions.id = clusters.region
9 INNER JOIN patterns ON clusters.pattern = patterns.pattern
10 INNER JOIN bmu ON patterns.pattern = bmu.pattern
11 INNER JOIN versions_mfiles ON bmu.id_version_mfile = versions_mfiles.id
12 INNER JOIN mfiles ON versions_mfiles.id_mfile = mfiles.id
13 INNER JOIN toolboxes ON mfiles.id_toolbox = toolboxes.id
14 WHERE
15     clusters.model = 100010
16 GROUP BY
17     region, neuron, toolbox, mfile_name

```

Na Listagem 6.12 está representada a *query* que nos diz quantos m-files estão associados a cada região. Na Tabela 6.4 os valores estão representados. Ao contrário do verificado na sub-seção 6.5.1, a área ocupada por cada retângulo de cor diferente é proporcional ao número de m-files associados a cada região. Na Tabela 6.4 estão representados o número de m-files por região que justificam a área ocupada por cada região no *TreeMap*. O número de resultados retornados pela *query* são a propriedade que define as áreas dos rectângulos de cada neurónio representado no *TreeMap*.

## 6.6 Vista do Módulo de Queries

A página do módulo de *queries* é constituída por um módulo de visualização de código, um módulo de m-files do neurónio e por um módulo de *queries*.

O único módulo novo a apresentar neste tipo de vista é o módulo de *queries*. Este módulo possui ferramentas para procura de dados na base de dados por parte dos utilizadores *Joës*. Este módulo realiza procuras de sequências de *tokens*.

As *queries* sequenciais de *tokens* permitem analisar o repositório *MATLAB* sem recorrer ao uso do *UbiSOM*. É o único tipo de análise que pode ser feito sobre os dados sem recorrer ao treino de SOMs.

The screenshot displays the MATRIXSTUDIO web application. On the left, a code editor shows the MATLAB script 'plots.m'. On the right, a search results table titled '101 Neuron Mfiles' and '32 Results of Search Patterns' is visible. The table lists search results with columns for 'Toolbox', 'Mfile', 'if LinePos', 'margin LinePos', and 'File'. Below the table, there is a search interface with input fields for 'if' and 'margin', and buttons for 'Search Patterns', 'Regions Queries', 'More Relevant', 'Continuous Search', 'Add tokens', and 'Add operators'.

Toolbox	Mfile	if LinePos	margin LinePos	File
/benchmarks-Mfile/td-vby/b...	script_run_profile.m	13   0	13   3	↗
/benchmarks-Mfile/td-vby/b...	plotsIfDescriptor.m	80   0	80   3	↗
/benchmarks-Mfile/td-vby/b...	plotsIfFrame.m	74   0	74   3	↗
/benchmarks-Mfile/td-vby/b...	plots.m	46   0	46   3	↗
/benchmarks-Mfile/td-vby/b...	sift_compile.m	11   0	11   3	↗
/benchmarks-Mfile/td-vby/b...	buildCopy.m	22   0	22   4	↗

Figura 6.13: Vista Módulo de Queries

### 6.6.1 Queries Sequências de Tokens

Este tipo de *queries* tem como funcionalidade procurar m-files presentes no repositório que tenham uma determinada sequência de *tokens*.

As *queries* para procura de sequencias de *tokens* são criadas durante a execução da aplicação. Para os diferentes tipos de *queries* sequenciais de *tokens* implementadas é utilizado o mesmo algoritmo para a construção das *queries*.

As *queries* são formadas através da concatenação de *strings* que constroem uma *string* final com a semântica correta associada à linguagem *SQL*, posteriormente a *string* é utilizada como *query* à base de dados. As *queries* possíveis de criar, em termos de estrutura, diferem apenas no *WHERE* da *query*.

A estrutura do corpo do *SELECT* e do *FROM* é igual para todos os tipos de *queries* sequencias. O *FROM* da *query* é um conjunto de junções naturais entre *subqueries* criadas por *token* selecionado para procura. Por cada *token* uma *subquery* é criada que será utilizada como tabela a consultar no *FROM* das *queries* sequenciais de *tokens*. A *subquery* está representada na Listagem 6.14.

Listagem 6.13: Código *PHP* que monta o *SELECT* da *query*

```

1  select = "SELECT
2  subQuery1.subMfile AS mfile ,
3  subQuery1.subToolName AS toolbox ,
4  subQuery1.subMfileName AS mfileName ,";
5
6  foreach($tokenSequence as $token){
7      $counter++;
8      $select .= 'subQuery' . $counter . '.subTokensName AS tokenName'
9      . $counter . ',';
10     $select .= 'subQuery' . $counter . '.subLine AS line'
11     . $counter . ',';
12     $select .= 'subQuery' . $counter . '.positionTok AS position'
13     . $counter . ',';
14
15     ...
16 }

```

Na Listagem 6.13 está representada a construção do *SELECT*. Os campos projetados pela *query* são:

- o id do m-file onde a sequência foi encontrada, o nome da toolbox do m-file e o nome do mesmo.
- Por cada *word token* ou *operator token* presente na sequência de *tokens*: o nome do *token*, o número da linha onde este está presente e a posição (número de coluna) da linha a que pertence.

Na listagem 6.14 está representada a construção do corpo do *FROM*.

Por cada *token* constituinte da sequência de *tokens* a procurar, uma *subquery* é gerada da relação das tabelas *mfiles*, *version\_mfiles*, *blocks\_mfiles*, *lines\_mfiles*, *lines\_tokens* e *tokens*.

A *subquery* serve para projetar os atributos necessários para relacionar as instâncias *tokens* para procurar sequências de *tokens* no repositório de dados. A tabela *toolbox* também contemplada na construção das *queries*, é utilizada para ir buscar informação complementar para representação no *frontend*.

O caso apresentado centra-se na procura das ocorrências de instâncias *word tokens* if *nargin* no mesmo m-file. A Listagem 6.14 é representação do algoritmo representado na Listagem 6.13. A variável *\$token* representada na Listagem seguinte assume os valores *if* e *nargin* no decorrer do algoritmo.

Listagem 6.14: Código PHP que monta o *FROM* da *query*

```

1 $counter++;
2
3 if($counter != 1){ $from .= ' INNER JOIN '; }
4
5 $from .= '(SELECT' .
6 'tl' . $counter . '.id AS subTool, tl' . $counter . '.name
7 AS subToolName,'
8 'm' . $counter . '.id AS subMfile, m' . $counter . '.name
9 AS subMfileName,'
10 't' . $counter . '.id AS subTokens, t' . $counter . '.name
11 AS subTokensName,'
12 'l' . $counter . '.line AS subLine, lt' . $counter . '.ci
13 AS positionTok'
14 . 'FROM'
15 . 'toolboxes tl' . $counter
16 . ' INNER JOIN mfiles m' . $counter . ' ON (tl' . $counter . '.id ='
17 . 'm' . $counter . '.id_toolbox)'
18 . ' INNER JOIN versions_mfiles vm' . $counter . ' ON (m' . $counter . '.id ='
19 . 'vm' . $counter . '.id_mfile)'
20 . ' INNER JOIN blocks_mfiles b' . $counter . ' ON (vm' . $counter . '.id ='
21 . 'b' . $counter . '.id_vers_mfile)'
22 . ' INNER JOIN lines_mfiles l' . $counter . ' ON (b' . $counter . '.id ='
23 . 'l' . $counter . '.id_block_mfile)'
24 . ' INNER JOIN lines_tokens lt' . $counter . ' ON (l' . $counter . '.id ='
25 . 'lt' . $counter . '.id_line)' .
26 . ' INNER JOIN tokens t' . $counter . ' ON (lt' . $counter . '.id_token ='
27 . 't' . $counter . '.id)'
28 . 'WHERE t' . $counter . '.name = "' . $token . '"') AS subQuery' . $counter;
29
30 array_push($tokens, $token);

```

O objetivo da *subquery* é encontrar os m-files em que existe pelo menos uma instância do *token* selecionado. As tabelas *toolboxes* e *mfiles* são utilizadas para saber o m-file e a toolbox em questão. É através da relação das tabelas que constituem um m-file que é possível saber se um m-file contém a instância *token* referida e as consequentes informações relacionadas com a sua posição no m-file.

Após a produção das *subqueries* temos uma tabela por *token* em que se sabe os m-files que contêm a instância *token* e a posição da mesma. Para garantir a sequência de *tokens* submetida no *WHERE* da *query* as posições das instâncias *token* encontradas são comparadas de forma a verificar que um m-file contém a sequência.

O *WHERE* destas *queries* assenta numa lógica de 2 tipos de condições que se têm de verificar. A segunda condição é uma conjunção de duas sub condições, que serão apresentadas como 2.1 e 2.2. Assim as condições são:

- **Condição 1:** as instâncias *tokens* encontradas têm de estar contidas no mesmo m-file. Assim, o id do m-file codificado em cada instância *token* é o mesmo, ver Listagem 6.15. Esta condição é a única que é comum aos diferentes tipos de *queries* sequenciais de *tokens*.

Listagem 6.15: Condição do id do m-file para cada instância *token* encontrada ser o mesmo

```
1 $cond1 .= '(subQuery' . $i . '.subMfile = subQuery' . $next . '.subMfile) AND ';
```

As outras duas condições são diferentes consoante o tipo de pesquisa requisitado, dessa forma os exemplos das condições são apresentados por tipo de *query*.

Uma das condições em cada um dos tipos de pesquisa visa a tratar os dados em que, duas instâncias *token* ordenadas da sequência estão na mesma linha e o caso em que as instâncias estão em linhas diferentes.

Tanto para a pesquisa normal como para a pesquisa mais relevante, a segunda condição do *WHERE* é igual, dessa forma no exemplo das condições, na pesquisa mais relevante, a segunda condição não será apresentada, visto que será apresentada no exemplo da pesquisa normal.

O tipo de pesquisa de sequência de *tokens* é escolhido no *frontend* através dos botões implementados no módulo de *queries*, ver Figura 6.13.

### Pesquisa Normal

A pesquisa normal de sequência de *tokens* tem em atenção a procura da sequência em que permite a possibilidade de haver outras instâncias *token* entre as instâncias *tokens* que constituem a sequência. Ou seja, devolve todos resultados em que existam as instâncias *token* ordenadas pela ordem da sequência de *tokens* submetidas, no mesmo m-file.

- **Condição 2.1:** duas instâncias *token* ordenadas da sequência estão na mesma linha e a posição da primeira instância *token* é menor que a posição da segunda instância *token*, ver Listagem 6.16, primeira ocorrência '*\$where*' representado.
- **Condição 2.2:** duas instâncias *token* ordenadas da sequência não estão na mesma linha, no entanto a linha da primeira instância *token* é inferior à linha da segunda instância *token*, ver Listagem 6.16, segunda ocorrência '*\$where*' representado.

Listagem 6.16: Código *PHP* que monta o *WHERE* da *query*

```
1 $where .= '(
```



```

2 | (
3 | (subQuery' . $i . '.subLine = subQuery' . $next . '.subLine)
4 | AND
5 | (subQuery' . $i . '.positionTok < subQuery' . $next . '.positionTok)
6 | ) OR ';
7 |
8 | $where .= '(subQuery' . $i . '.subLine < subQuery' . $next . '.subLine)
9 | ) AND ';

```

### Pesquisa mais relevante

A pesquisa mais relevante impõe um limite de diferença do número de linhas entre a ocorrência das instâncias *token* submetidas na sequência de *tokens*.

- **Condição 2.1:** duas instâncias *token* ordenadas da sequência não estão na mesma linha, no entanto a linha da primeira instância *token* é inferior à linha da segunda instância *token* e a diferença entre o número linhas é inferior ou igual a um valor (\$numberOfLines), ver Listagem 6.17.

Listagem 6.17: Código *PHP* que monta o *WHERE* da *query*

```

1 | $where .= '(subQuery' . $i . '.subLine < subQuery' . $next . '.subLine)
2 | AND
3 | (subQuery' . $next . '.subLine - subQuery' . $i . '.subLine <= ' . $numberOfLines . ')
4 | ) AND ';

```

### Pesquisa Contígua

- **Condição 2.2:** duas instâncias *token* ordenadas da sequência estão na mesma linha e a posição da primeira instância *token* é menor que a posição da segunda instância *token*.



## CONCLUSÃO

O trabalho realizado resultou do desenvolvimento de um protótipo *web* que possibilita a visualização de código *MATLAB*, análise a nível de expressões de tokens (módulo de queries sequencias) e analisar o modelo SOM estudado em trabalho anterior [26] através de metáforas visuais. Realizou-se um estudo de metáforas visuais que tinham potencial para representar os dados produzidos pelo SOM de forma a que um utilizador do protótipo conseguisse retirar conclusões sobre os mesmos. Com a realização desta dissertação conseguiu-se transformar o repositório *MATLAB* num repositório *web*, um utilizador *Joe* (não especialista, ver início do capítulo 1) consegue visualizar código *MATLAB* proveniente do repositório *web*, procurar sequências de tokens presentes no repositório *web*, ver secção 6.6.1 e visualizar *TreeMaps* representativos do repositório *web* em relação ao estudo dos *concerns* e *TreeMaps* sobre o modelo SOM estudado em trabalhos anteriores [26]. O utilizador *Stephanie* (especialista, ver início do capítulo 1) pode anotar linhas e instâncias tokens de m-files e produzir metáforas visuais para *Joes*.

O desenho da base de dados que alimenta o protótipo, permitiu que identificássemos que tipos de dados precisávamos de produzir sobre os m-files e sobre o modelo SOM estudado [26] para alimentar a mesma. A estrutura da base de dados foi um processo incremental no sentido que ao longo do desenvolvimento de todo o projeto esta foi modificada para suportar funcionalidades não contempladas inicialmente.

Estendeu-se a ferramenta *CCCExplorer* no sentido de tornar o processo de importação de repositórios *MATLAB* o mais autónomo através da produção, realizada pelo *CCCExplorer*, do *output SQL* de toolboxes. A extensão realizada, permite transformar repositórios de grandes dimensões automaticamente em questão de minutos para importação na base de dados e dessa forma alimentar as análises implementadas no protótipo.

Pensou-se no módulo de *queries* sequencias devido à necessidade de encontrar casos específicos de ocorrências simultânea entre dois ou mais instâncias tokens num m-file. Notámos que armazenando todos os tokens dos m-files na base de dados, dava-nos uma ferramenta poderosa de consulta dentro de cada m-file. Durante a extensão do *CCCExplorer* introduzimos a propriedade de localização de uma instância token no m-file (número de linha e coluna inicial da instância token, ver secção 6.2.1). A implementação desta ideia abriu portas para uma análise suplementar do protótipo, permitindo um estudo do código *MATLAB* segundo uma perspectiva diferente do SOM utilizando por base os mesmos dados, contribuindo dessa forma para o estudo da modularidade no código *MATLAB*. Através deste tipo de pesquisa, estudos futuros podem ser realizados de forma a procurar casos típicos de código *MATLAB* onde se verifica fenómenos de *crosscutting concerns*, e dessa forma estudar e implementar alterações ao código para evitar tais casos. Outros tipos de pesquisas e podem ser contempladas na análise de modelos relacionais de modelos SOM. Tais módulos poderão considerar a procura da listagem dos m-files mais significativos como a procura dos m-files presentes em regiões identificadas em modelos SOM.

Com a implementação do *TreeMap* obtivemos uma análise global do repositório. Verificou-se que através da visualização de *TreeMaps* um utilizador *Joe* conseguia concluir informações tais como, m-files mais significativos presentes no repositório e densidade dos *concerns* presentes no repositório de código, que *Stephanies* foram capazes de validar através do uso de *queries* à base de dados. Tendo em conta o *TreeMap* referente ao modelo SOM estudado [26], apenas um estudo preliminar foi realizado. O que se verificou, pelas Figuras 6.8 e 6.9, que demasiada informação era representada nos *TreeMaps*. Dessa forma, em trabalho futuro, devem-se criar filtros, de forma a refinar o conjunto de dados a representar em *TreeMaps* para que uma melhor representação dos dados seja possível. Esta melhoria permitiria uma melhoria substancial na análise da metáfora visual. Possíveis filtros poderão ser disponibilizados aos utilizadores *Joels* para que eles próprios possam filtrar o conjunto de dados e dessa forma potenciar a análise da representação por parte dos utilizadores. A utilização de gradientes nas cores utilizadas nos *TreeMaps* pode auxiliar na distinção de elementos do mesmo nível de hierarquia dos dados.

## BIBLIOGRAFIA

- [1] B. Barroso. *O poder das metáforas visuais* | Bruno Barroso | Pulse | LinkedIn. <https://www.linkedin.com/pulse/20140612142825-5996819-o-poder-das-metáforas-visuais>. Consultado: Fev. de 2018.
- [2] bootstrap. *Bootstrap · The most popular HTML, CSS, and JS library in the world*. <https://getbootstrap.com/>. Consultado: Fev. de 2019.
- [3] M. Bostock. *D3.js - Data-Driven Documents*. <https://d3js.org/>. Consultado: Fev. de 2018.
- [4] M. Bruntink, A. Van Deursen, T. Tourwe e R. van Engelen. “An evaluation of clone detection techniques for crosscutting concerns”. Em: *20th IEEE International Conference on Software Maintenance, 2004. Proceedings*. IEEE. 2004, pp. 200–209.
- [5] G. d. F. Carneiro. “SourceMiner: Um Ambiente Integrado Para Visualização Multi-Perspectiva De Software”. Em: (2013).
- [6] T. D. V. Catalogue. *Treemap - Learn about this chart and tools to create it*. <https://datavizcatalogue.com/methods/treemap.html>. Consultado: Fev. de 2018.
- [7] T. D. V. Catalogue. *What is a Treemap?* [https://docs.tibco.com/pub/spotfire/6.5.0/doc/html/images/tree\\_example\\_three\\_levels.png](https://docs.tibco.com/pub/spotfire/6.5.0/doc/html/images/tree_example_three_levels.png). Consultado: Fev. de 2018.
- [8] T. D. V. Catalogue. *What is a Treemap?* [https://docs.tibco.com/pub/spotfire/6.5.0/doc/html/tree/tree\\_what\\_is\\_a\\_treemap.htm](https://docs.tibco.com/pub/spotfire/6.5.0/doc/html/tree/tree_what_is_a_treemap.htm). Consultado: Fev. de 2018.
- [9] N. Cavaleiro Marques, M. Monteiro e B. Silva. “Analysis of a token density metric for concern detection in Matlab sources using UbiSOM”. Em: *Expert Systems* 35.4 (2018), e12306.
- [10] cs.toronto.edu. *Data Manipulation Language and Data Query Language*. <http://www.cs.toronto.edu/~nn/csc309-20085/guide/pointbase/docs/html/htmlfiles/dmlDql.html>. Consultado: Fev. de 2019.
- [11] E. W. Dijkstra. “On the role of scientific thought”. Em: *Selected writings on computing: a personal perspective*. Springer, 1982, pp. 60–66.
- [12] K. I. P. Duarte. “Limitations in the Support to Modularity in MATLAB: a Survey-based Empirical Study”. Tese de doutoramento. FCT-UNL, 2017.

- [13] S. G. Eick e A. F. Karr. “Visual scalability”. Em: *Journal of Computational and Graphical Statistics* 11.1 (2002), pp. 22–43.
- [14] D. A. Keim. “Information visualization and visual data mining”. Em: *IEEE transactions on Visualization and Computer Graphics* 8.1 (2002), pp. 1–8.
- [15] D. A. Keim e H.-P. Kriegel. “Visualization techniques for mining large databases: A comparison”. Em: *IEEE Transactions on knowledge and data engineering* 8.6 (1996), pp. 923–938.
- [16] A. Kellens, K. Mens e P. Tonella. “A survey of automated code-level aspect mining techniques”. Em: *Transactions on aspect-oriented software development IV*. Springer, 2007, pp. 143–162.
- [17] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier e J. Irwin. “Aspect-oriented programming”. Em: *European conference on object-oriented programming*. Springer. 1997, pp. 220–242.
- [19] O que é Laravel? Porque usá-lo? João Roberto. <https://medium.com/\spacefactor\@m\joaorobertopb/o-que-é-laravel-porque-usá-lo-955c95d2453d>. Consultado: Fev. de 2018.
- [20] M. Malhotra. *Why Laravel Is The Best PHP Framework In 2018?* <https://www.valuecoders.com/blog/technology-and-apps/laravel-best-php-framework-2017>. Consultado: Fev. de 2018.
- [21] N. C. Marques, B. Silva e H. Santos. “An Interactive Interface for Multi-dimensional Data Stream Analysis”. Em: *2016 20th International Conference Information Visualization (IV)*. 2016, pp. 223–229. DOI: 10.1109/IV.2016.72.
- [22] MathWorks. *Unsupervised Learning*. <https://www.mathworks.com/discovery/unsupervised-learning.html>. Consultado: Fev. de 2018.
- [23] D. Matos, N. C. Marques e M. G.M. S. Cardoso. “Stock market series analysis using self-organizing maps”. Em: *Revista de Ciências da Computação* 9.9 (2015).
- [24] M Monteiro, J Cardoso e S. Posea. “Identification and characterization of crosscutting concerns in MATLAB systems”. Em: *Conference on Compilers, Programming Languages, Related Technologies and Applications (CoRTA 2010), Braga, Portugal*. CiteSeer. 2010.
- [25] M. P. Monteiro e J. M. Fernandes. “Towards a catalogue of refactorings and code smells for AspectJ”. Em: *Transactions on aspect-oriented software development I*. Springer, 2006, pp. 214–258.
- [26] M. P. Monteiro, N. C. Marques, B. Silva, B. Palma e J. Cardoso. “Toward a Token-Based Approach to Concern Detection in MATLAB Sources”. Em: 2017, pp. 3–4.
- [27] S. Mukherjea e J. D. Foley. “Requirements and Architecture of an Information Visualization Tool”. Em: *Workshop on Database Issues for Data Visualization*. Springer. 1995, pp. 57–75.

- 
- [28] MySQL. *MySQL :: MySQL 8.0 Reference Manual :: 1 General Information*. <https://dev.mysql.com/doc/refman/8.0/en/introduction.html>. Consultado: Fev. de 2019.
- [29] MySQL. *MySQL :: MySQL 8.0 Reference Manual :: 1.3.2 The Main Features of MySQL*. <https://dev.mysql.com/doc/refman/8.0/en/features.html>. Consultado: Fev. de 2019.
- [30] T. Otwell. *Laravel - The PHP Framework For Web Artisans*. <https://laravel.com/>. Consultado: Out. de 2017.
- [31] U. Propel. *Data Visualization and D3.js | Udacity*. <https://in.udacity.com/course/data-visualization-and-d3js--ud507>. Consultado: Fev. de 2018.
- [32] B. M. dos Santos Palma. "A Tool for Mining Concerns in MATLAB Code Repositories". Tese de doutoramento. FCT-UNL, 2017.
- [33] SearchSQLServer. *What is database (DB)? - Definition from WhatIs.com*. <https://searchsqlserver.techtarget.com/definition/database>. Consultado: Out. de 2018.
- [34] M. Shaw. "Modularity for the Modern World: Summary of Invited Keynote". Em: *Proceedings of the Tenth International Conference on Aspect-oriented Software Development*. AOSD '11. Porto de Galinhas, Brazil: ACM, 2011, pp. 1–6. ISBN: 978-1-4503-0605-8. DOI: 10.1145/1960275.1960277. URL: <http://doi.acm.org/10.1145/1960275.1960277>.
- [35] A. Silberschatz, H. F. Korth e S. Sudarshan. *Database system concepts*. 6ª ed. New York: McGraw-Hill, 2010. URL: <http://www.db-book.com/>.
- [36] A. N. Silva, G. Carneiro, R. Zanin, A. Dal Poz e E. Martins. "Propondo uma Arquitetura para Ambientes Interativos baseados em Múltiplas Visões". Em: *II Workshop Brasileiro de Visualização de Software*. 2012, pp. 1–8.
- [37] B. Silva e N. C. Marques. "The ubiquitous self-organizing map for non-stationary data streams". Em: *Journal of Big Data* 2.1 (2015).
- [38] R. Spence. *Information Visualization: Design for Interaction*. Pearson/Prentice Hall, 2007. ISBN: 9780132065504. URL: <https://books.google.pt/books?id=bKQeAQAAIAAJ>.
- [39] TechTerms. *Token Definition*. <https://techterms.com/definition/token>. Consultado: Fev. de 2019.
- [40] tutorialspoint. *MATLAB M-Files*. [https://www.tutorialspoint.com/matlab/matlab\\_m\\_files.htm](https://www.tutorialspoint.com/matlab/matlab_m_files.htm). Consultado: Fev. de 2019.
- [41] w3schools. *SQL Tutorial*. <https://www.w3schools.com/sql/>. Consultado: Fev. de 2019.

- [42] C. Ware. “Visual Queries: The Foundation of Visual Thinking”. Em: *Knowledge and Information Visualization: Searching for Synergies*. Ed. por S.-O. Tergan e T. Keller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 27–35. ISBN: 978-3-540-31962-7. DOI: [10.1007/11510154\\_2](https://doi.org/10.1007/11510154_2). URL: [https://doi.org/10.1007/11510154\\_2](https://doi.org/10.1007/11510154_2).
- [43] Wikipédia. *Treemapping* – Wikipédia, a enciclopédia livre. <https://pt.wikipedia.org/wiki/Treemapping>. Consultado: Fev. de 2018.
- [44] Wikipedia. *D3.js*. [https://en.wikipedia.org/wiki/D3.js#cite\\_note-Viau\\_2012/06-3](https://en.wikipedia.org/wiki/D3.js#cite_note-Viau_2012/06-3). Consultado: Fev. de 2018.